Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

March 23, 1992

••••Last Minute Notes on the Update ••••

MacApp V3 Users

There were important changes to Tlist_Redef and USIJPascalObject.p so make sure to get the new versions and:
1) rebuild SIJ.Lib, rebuild your appplication
2) copy the new type definitions into your ".dsi' file in place of the old ones.
Note that Tlist_Redef lives in the :MPW:Scripts: folder.

All Users

In order to squeeze a revised copy of the Debugger Tech Notes (DTNs) on the disk I omitted the 'Rom .snt for IIci/IIfx/IIsi' folder.
You may wish to add it to the '3/22/92 Dbgr_upd' folder on your hard disk after you unpack the '.sea' file.

The reverse side of this sheet is a description of *CoverTest*, a new program that I have been working on. Old users must re-up (send money) to get a copy of it when it ships in June (of this year).

A bit of David Letterman style Humor from some friends at Apple

THE UNOFFICIAL TOP TEN REASONS TO PROGRAM FOR WINDOWS

10. I want to beat Claris to the punch
9. Donald Trump is doing Windows programming to make ends meet
8. I'll be more productive with Hungarian notation
7. C++ isn't enough of a challenge by itself
6. Bill Gates has far more money than John Sculley
5. We can finally feel superior to the competition
4. To remind us we actually like MPW
3. I secretly like MacApp 1.1 better
2. I've always wanted a square, white cursor

And the number one reason to program in windows is...

1. My application doesn't need to scroll

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

October 4, 1992

Dear Mac II Nosy/Debugger User,

Enclosed is the latest release of the Universal Version of *MacNosy* and *The Debugger V2/92*, and some accompanying notes on them. This release contains bug fixes and new features.

Tech Support for The Debugger is available via phone, Applelink (D1037), or CompuServe.

Site (multiple copy) Licenses are available through Jasik Designs. The pricing is $225 per copy times the number of copies (programmers using the product) plus sales tax and $20 shipping.

68040 Notes

The Jump trace bit does not work correctly in the 68040 CPU chips (Motorola screwed up), and as a consequence, the 'Trace Jumps' and 'Proc Entry/Exit Trace' commands do NOT work on machines with these CPU's. **Motorola DOES NOT intend to fix the problem. Because of this and the fact that MMU protection works better on the 68030 Macs (IIci, IIfx), they are still the best machines for debugging.**

INIT notes

MicroSoft Mail must load **after** *The Debugger*.
SuperBoomerang- use V3.01 or later as some of the 3.0 versions had bugs.
After Dark users should be using version 2.0V or later. It should load **after** *The Debugger*.
xDbgr_Startup **MUST** be placed in the Extensions folder on System 7.
NOW utilities Startup Mgr - **disable the options**: 'protect screen from erasure' & 'display invisible INITs'
IT_ADB is an INIT to activate the 'interrupt' (Command-power) key on the IIsi and Mac LC machines.
It is in the folder: '`Misc INITs & Source`'.

Installation Procedure

This update is shipped as a Self Extracting Archive (.sea) file. Double click on it to unpack the files.
The ROM/CODE.snt for the Mac II/IIx/IIcx/SE30 **was omitted**, get a copy from an older release.
If you have MPW, then I strongly suggest that you use the 'Install Debugger' script .
To manually install *The Debugger*, do the following:
Copy the files in the "put files in System fldr" to your System folder. **You must do this step.**
Copy or create a ROM .snt (default in 'Dbgr/Nosy files' is for a Mac IIci/IIfx/IIsi, Quadra, PowerBook 1xx ...)
Boot the mac, and verify that the new version is operational.
There are **no significant changes to the MacApp or IBS mods** in this release. You may omit changing them.
If you are using Version 3 of MacApp, then merge the IBS mods into the IBS folder.
If you are using IBS, then reinstall it, using the IBS_Install script.

System Compatibility

The Debugger and *MacNosy* have been tested on Systems 6.0.5, 6.0.7, 7.0 , 7.0.1, 7.1, & the System 7 TuneUp.
For the Classic and LC, you must build a ROM/CODE.snt file from scratch.

Miscellaneous Notes

• You may now **change command key equivalents** in The Debugger via the '`=Menu`' command in the .dsi file, see the documentation in ROM.dsi for details. Now you can close windows with Cmd-W !!

If you are using **7.1 and the Asian language extensions**, then you will have to contact Mark Zeren at Apple to get a patch so that The Debugger will work (Applelink id is "ZEREN"). You should also change the Cmd-space menu equivalent to Cmd-tab. Activate the commented line in the ROM.dsi in the '=M' section.

...ave been successful in shipping patches to The Debugger to some of you this spring via AppleLink, and I will continue to do so this fall. I can NOT send binary files across to Internet addresses.
Send me your AppleLink ID if you haven't already done so.
Patch files sent out this fall will use this version (10/4/92) as the base version.

Debugger Tech Notes (DTN's) are on the floppy in MicroSoft Word 4 format along with a table of contents.

OBJECT MASTER

What is Object Master?

Object Master is a tool designed for programmers using MPW (C, C++, and Pascal), THINK Object Pascal and C, Modula 2 - P1, and other programming languages on the Macintosh. Object Master greatly increases programmer productivity by providing intelligent source code editing, object browsing, resource browsing, segmentation mapping, and file mapping all in an environment fully integrated with the above programming languages via Apple Events.

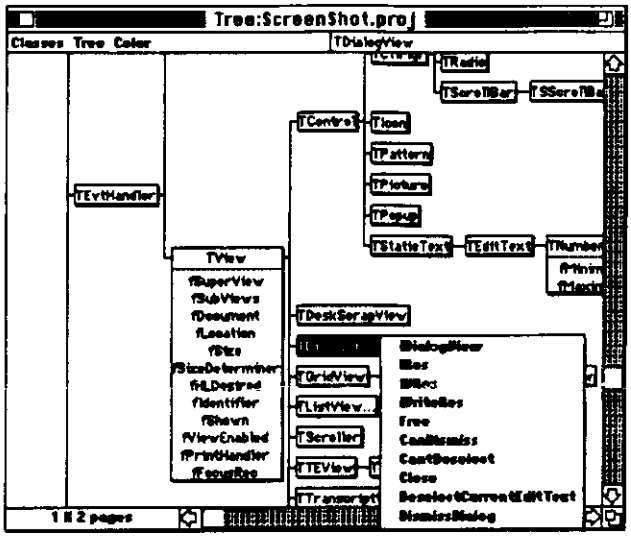
Intelligent source code editing

One example of intelligent source code editing is the ability of the Object Master editor to check syntax on-the-fly, catching most errors without having to compile. As you create your files, a marker list is automatically generated and provides easy access to all procedures and objects that have been defined in the file.

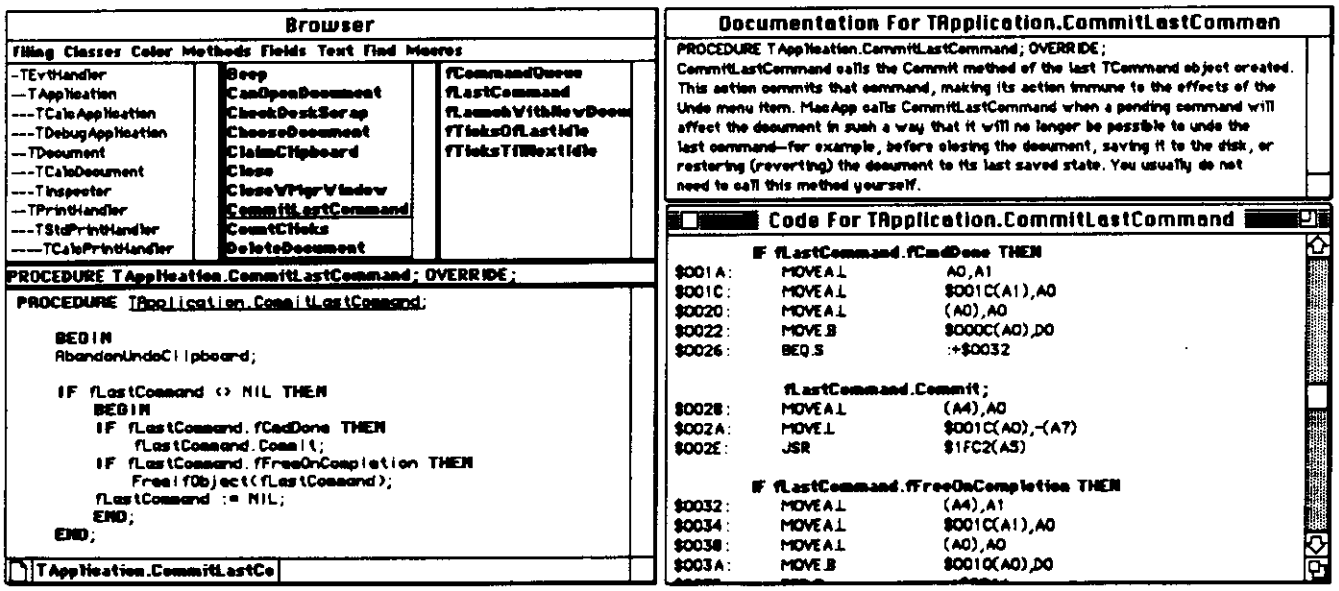
With Object Master, you can define custom code templates that are used to check the consistency of your personal coding style. The Object Master editor also allows you to assign unique colors and styles to procedures, keywords, and templates. Automatic indenting is also supported by Object Master.

Organize your files into a project

Object Master automatically maintains a database of data definitions and procedures. This organization of your files provides for fast navigation and structure editing. An Object Master project can mix Pascal, C, Modula, Rez, and 411 files. For example, a Pascal program that calls several C libraries can be edited as a single project with Object Master.



Class Tree Window provides graphical display and navigation through the project's class hierarchy.



A Documentation Window and a Code Display Window can be linked to a Browser Window to provide automatic access to any 411 documentation for the method being edited as well as a disassembled view of the method.

Develop Applications Quickly

OOPC is a high-level toolkit for creating professional object-oriented Macintosh applications quickly and easily.

with

OOPC™

OOPC turns ANSI-standard C into a powerful, object-oriented programming language; power far beyond C++, but much easier to learn and use.

A Complete Class Library For Easy Application Development

OOPC has automatic document handling. With the OOPC class library, a document automatically displays itself in a window, manages offscreen bitmap memory, can save itself to file and restore itself, print itself, handle a user request to cut, copy, clear, or paste objects, or undo an action. You don't have to write a single line of code to manage documents.

OOPC has a streamlined class architecture. An actor, such as a document, manages a window. A window knows which views are visible in the window. A view keeps track of the objects in the view. This logical hierarchy explains both how objects are displayed and how objects are acted upon. This simple approach is efficient, and makes it easy to follow what's going on.

OOPC has complete support for an application user interface. You can expect any class library to let you create and handle windows, menus, dialogs, and so on. OOPC gives you tear-off menus, floating windows, plus tool, pattern and color palettes. Pop-up menus are a standard control. All standard File and Edit menu items automatically work and are enabled/disabled for you. All modal dialogs you create are automatically movable modal dialogs. To give you an idea of how simple it is to use OOPC, visual operation of all controls and other items in a dialog is automatic; you can create, show, and handle a dialog in two lines of code. OOPC gives you more user interface features, and makes them easier to use.

OOPC includes an object-oriented graphics package. Plus, the graphics package is built upon a ***paint package***. OOPC also has ***styled text edit*** that works on any Macintosh in the world.

(over please; continued)

A Dynamic Object System For C With Awesome Power

OOPC has an object system that meets the theoretical boundaries of the power inherent in object orientation: multiple inheritance, flexible class-object data declaration, dynamic class definition, and method dispatch control. ***Because OOPC uses standard C, this power is packaged in a way that is easy to learn to use, and lets you take full advantage of object-oriented power simply and easily.***

Multiple inheritance gives you design flexibility by allowing inherited behavior from more than one class. With flexible class-object data declaration, a class may have its own data structure, separate from the objects it creates, and an object may have its own methods, different from (or in addition to) those specified by its class. You might want a class to hold data to be used as a common information pool for other classes and objects. Or you might want to define methods for objects, to further refine their behavior. When a user selects a menu item, for example, a specific menu item object method could be called to handle that selection.

OOPC is an object system that is fully dynamic: everything happens at run-time. Inheritance and methods can be changed at any time. This gives you great flexibility in letting classes and objects adapt to a changing environment.

A method is simply a function for a class. Method dispatch control means you can access multiple methods in a single call in whatever order you need. ***Dispatch control maximizes code use with minimum overhead. C++ can't touch this.***

OOPC's flexibility means you can construct software building blocks the way you want to get the job done your way. No limits.

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

March 23, 1992

Dear Mac II Nosy/Debugger User,

Enclosed is the latest release of the Universal Version of *MacNosy* and *The Debugger V2/92*, and some accompanying notes on them. This release contains numerous bug fixes and new features.

For those of you who purchased *The Debugger* prior to January 1, 1992, this is you last free update. To receive updates and support in 1992, fill out the enclosed invoice and return it to me (before May 1, 1992) with a check for $130 (payable to a US Bank) or authorization to bill your credit card.

Tech Support for The Debugger is available via phone, Applelink (D1037) or CompuServe.

Site (multiple copy) Licenses are available through Jasik Designs. The pricing is $225 per copy times the number of copies (programmers using the product) Plus sales tax and $20 shipping.

General

Site Licensee's - The floppy contains a stuffed copy of this cover letter in LinkSaver format.
Is your address label correct?? There is a file on the disk containing all the AppleLink addresses I have in my database. If yours is not in it, then please send me a link with your name.
Is The Debugger/MacNosy meeting your needs, if not complain to me.

68040 Notes

Quadra 700 & 900's may be speeded up by changing the 50 Mhz CPU clock Crystal to a 66 Mhz one yielding a 15 to 20% improvement in performance. The Jump trace bit does not work correctly in the 25Mhz 68040 CPU chips (Motorola screwed up), and as a consequence, the ' Trace Jumps' and 'Proc Entry/Exit Trace' commands do NOT work machines with these CPU's.

INIT notes

QuickTime must load **before** *The Debugger*, you can rename it to *aQuickTime*.
MicroSoft Mail must load **after** *The Debugger*.
SuperBoomerang- use V3.01 or later as some of the 3.0 versions had bugs.
After Dark users should be using version 2.0V or later, it should load **after** *The Debugger*.
xDbgr_Startup **MUST** be placed in the Extensions folder on System 7.
NOW utilities Startup Mgr causes problems and should be avoided. use Apple's extensions manager.
INIT_ADB is an INIT to activate the 'interrupt' (Command-power) key on the IIsi and Mac LC machines. It is in the folder: '`Misc INITs & Source`'.

Installation Procedure

This update is shipped as a Self Extracting Archive (.sea) file. Double click on it to unpack the files.
The ROM/CODE.snt for the Mac IIci/IIfx,... **was omitted**, get a copy from an older release.
If you have MPW, then I strongly suggest that you use the 'Install Debugger' script .
To manually install *The Debugger*, do the following:
Copy the files in the "put files in System fldr" to your System folder. **You must do this step.**
Copy or create a ROM .snt for your machine (default in Dbgr/Nosy files is for a 256K ROM Mac II)
Boot the mac, and validate that the new version is operational.
If you are using Version 3 of MacApp, then merge the IBS mods into the IBS folder.
If you are using IBS, then reinstall it, using the IBS_Install script.

System Compatibility

The Debugger and *MacNosy* have been tested on Systems 6.0.5, 6.0.7 7.0 , 7.0.1, 7.1, & the System 7 TuneUp. For the Classic and LC you must build a ROM/CODE.snt file from scratch.

Miscellaneous Notes

• You may now **change command key equivalents** in The Debugger via the '`=Menu`' command in the .dsi file, see the documentation in ROM.dsi for details. Now you can close windows with Cmd-W !!

OBJECT MASTER

What is Object Master?

Object Master is a tool designed for programmers using MPW (C, C++, and Pascal), THINK Object Pascal and C, Modula 2 - P1, and other programming languages on the Macintosh. Object Master greatly increases programmer productivity by providing intelligent source code editing, object browsing, resource browsing, segmentation mapping, and file mapping all in an environment fully integrated with the above programming languages via Apple Events.

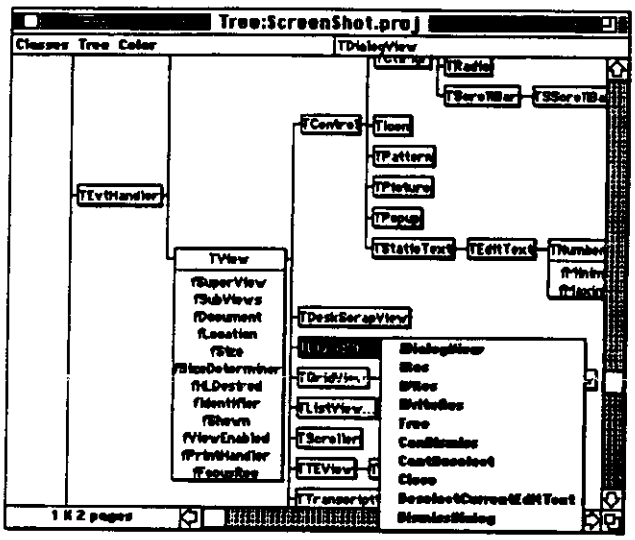
Intelligent source code editing

One example of intelligent source code editing is the ability of the Object Master editor to check syntax on-the-fly, catching most errors without having to compile. As you create your files, a marker list is automatically generated and provides easy access to all procedures and objects that have been defined in the file.

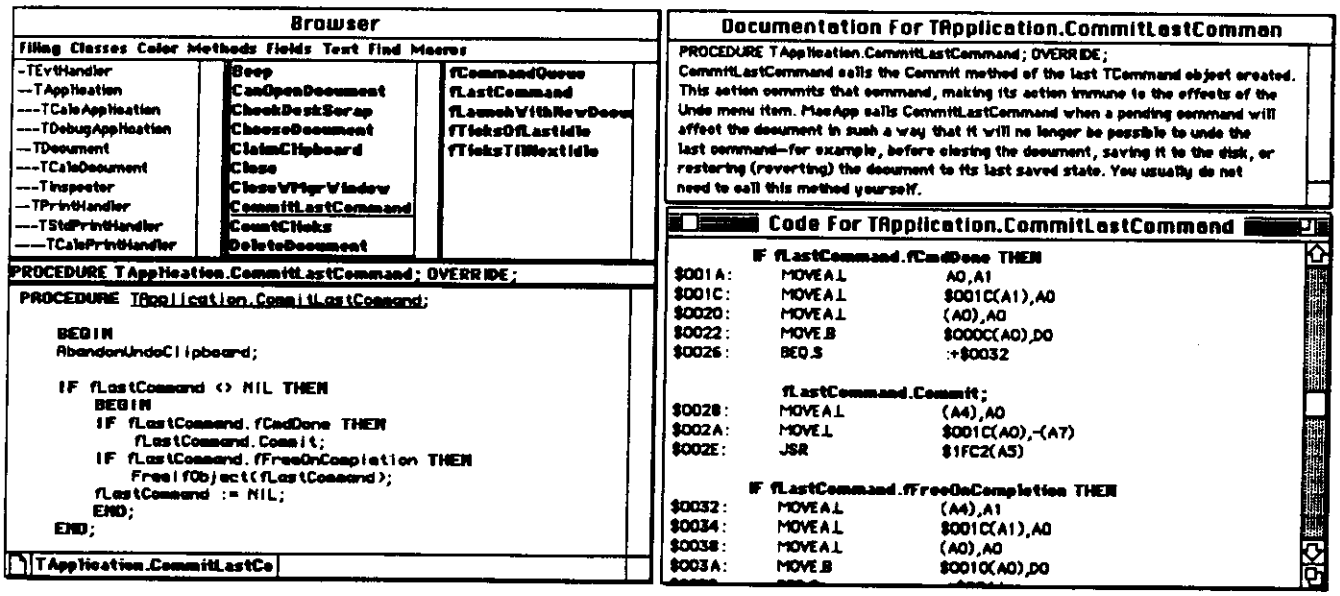
With Object Master, you can define custom code templates that are used to check the consistency of your personal coding style. The Object Master editor also allows you to assign unique colors and styles to procedures, keywords, and templates. Automatic indenting is also supported by Object Master.

Organize your files into a project

Object Master automatically maintains a database of data definitions and procedures. This organization of your files provides for fast navigation and structure editing. An Object Master project can mix Pascal, C, Modula, Rez, and 411 files. For example, a Pascal program that calls several C libraries can be edited as a single project with Object Master.



Class Tree Window provides graphical display and navigation through the project's class hierarchy.



A Documentation Window and a Code Display Window can be linked to a Browser Window to provide automatic access to any 411 documentation for the method being edited as well as a disassembled view of the method.

Invoice

March 24, 1992

From: Jasik Designs, 343 Trenton Way, Menlo Park, CA 94025
(415) 322-1386

To: paste your mailing label here

New Address _____
_____
_____

For:

Quantity	Item	Unit price	Total
_____	**Debugger/Nosy Updates in 1992**	**$130**	_____
_____	Additional Licenses for Univ Debugger	**$225**	_____
	Sales Tax @ 8.2 % for California Residents		_____
		Total	_____

If you are paying by Visa/MC then fill out:

Card Name: ____ VISA ___ MasterCard

Card Number: _____ Expiration: ___/___

Name of Cardholder (Print) _____

Your Signature: _____

Phone (in case of problems) _____ AppleLink # _____

Orders paid by credit card may be AppleLinked to me at D1037. Please include your card number, expiration, name, and authorization to debit your card by the stated amount.

Future updates may be supplied on 1.4 Meg floppies, please answer :
I have a Macintosh that has SuperDrive (1.4 Meg floppy) Yes___ NO___

Payment must accompany this form !!!

Orders paid for by Visa/MasterCard will show up on your Visa/MC statement as a charge from **Jasik Designs**. Paper receipts will be supplied on request only.

You are being billed for update fees for your copies of *The Debugger V2 & Nosy* as my records show that you purchased/updated your copies of it prior to Jan 1, 1992.

If you feel that I am in error, then please submit a copy of canceled checks or other proof of purchase that will show my error.

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

Nov 19, 1991

Dear Mac II Nosy/Debugger User,

Enclosed is the latest release of the Universal Version of *The Debugger V2/91 and MacNosy*, and some accompanying notes on them. This release contains numerous bug fixes and new features which are summarized in the Change History file. The next update will be mailed in Jan or Feb of 1992.
I will be at the Apple Tools exhibit at MacWorld in San Francisco (Jan 12 - 15, 1992).

Tech Support for The Debugger is available via phone, Applelink (D1037) or CompuServe.

Site (multiple copy) Licenses are available through Jasik Designs. The pricing is $225 per copy times the number of copies (programmers using the product) Plus sales tax and $20 shipping.

General

Site LIcensee's - The floppy contains a stuffed copy of this cover letter in LinkSaver format.
Is your address label correct?? There is a file on the disk containing all the AppleLink addresses I have in my database. If yours is not in it, then please send me a link with your name.
Is The Debugger/MacNosy meeting your needs, if not complain to me.

The Need for Speed

I am in the process of evaluating hardware speedups for the IIfx and the Quadra's. It appears that the Quadra's clock can be increased to 33 Mhz, yielding a 20% improvement in compile times.

INIT notes

QuickTime must be loaded **before** *The Debugger*, you can rename it to *aQuickTime*.
MicroSoft Mail must load **after** *The Debugger*.
Some versions of *SuperBoomerang 3.0* are buggy and will NOT work with *The Debugger*.
I am using *SuperBoomerang* 3.01ß3 without any problems.
After Dark users should be using version 2.0V or later, it should load **after** *The Debugger*.
xDbgr_Startup **MUST** be placed in the Extensions folder on System 7.
NOW utilities Startup Mgr causes problems and should be avoided. use Apple's extensions manager.
INIT_ADB is an INIT to activate the 'interrupt' (Command-power) key on the IIsi and Mac LC machines. It is in the folder: '`Misc INITs & Source`'.

Installation Procedure

This update is shipped as a Self Extracting Archive (.sea) file. Double click on it to unpack the files.
If you have MPW, then I strongly suggest that you use the 'Install Debugger' script .
To manually install *The Debugger*, do the following:
Copy the files in the "put files in System fldr" to your System folder. **You must do this step.**
Copy or create a ROM .snt for your machine (default in Dbgr/Nosy files is for a 256K ROM Mac II)
Boot the mac, and validate that the new version is operational.
If you are using Version 3 of MacApp, then merge the IBS mods into the IBS folder.
If you are using IBS, then reinstall it, using the IBS_Install script.

System Compatibility - Recommended System Level

This version of *The Debugger* and *MacNosy* has been tested on Systems 6.0.5, 6.0.7 7.0 & 7.0.1.
For the Classic and LC you must build a ROM/CODE.snt file from scratch.

Miscellaneous Notes

- I fixed the problem of *The Debugger* destroying your custom desktop pattern selection.
To install a special color desktop pattern in *The Debugger*, paste a 'ppat' into it in ResEdit.
- If you want to be able to change bit depths while running, then paste the resource in the folder: "Kevin's Scrn Mode Swapper" into *The Debugger* in ResEdit. (Thank you Kevin Mitchell).
- The ability to do a Step Continuous without flashing the screen no longer relies on the MMU, and is operational for all Mac II class machines. Use the `Remap_Scrn` debugger flag to toggle it.

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

August 3, 1991

Dear Mac II Nosy/Debugger User,

Enclosed is the latest release of the Universal Version of *MacNosy* and *The Debugger V2/91*, and some accompanying notes on them. This release contains numerous bug fixes and new features which are summarized in the Change History file on the update disk.
The next update will be mailed out around November 1, 1991.

Tech Support for The Debugger is available via phone, Applelink (D1037) or CompuServe.

Site (multiple copy) Licenses are available through Jasik Designs. The pricing is $225 per copy times the number of copies (programmers using the product) Plus sales tax and $20 shipping.

When you are up to your ass in alligators, you forget that the original goal was to drain the swamp

Over the last couple of months, I have been fighting increasing entropy (disorder) in my programs, the Mac, etc, and losing the battle. Some of the problems (bugs) that you have reported to me have not been fixed in this release or got lost. You may want to resubmit your complaints.

Maxima V2 for System 7 is available from Connectix. Phone 800-950-5880 or AppleLink (connectix) .

Object Master is a new language sensitive editor from Acius. The enclosed literature describes it.

Quick Keys & The Debugger on System 7

Quick Keys Version 2.1 fixes some incompatibilities between it and The Debugger. A later version should be able to "load" before The Debugger so it can be used inside it.

Zortech C++ & The Debugger

The last version of Zortech C++ that I saw (2.1r2) was improved, but still had some minor problems with the format of the .SYM file output of the type info for Source level debugging.

Debugger Tech Notes #9 & 10

discusses some problems with MPW 3.2 release on ETO #4. A copy of the Runtime.o that will appear on ETO #5 is on the disk. DTN #10 discusses the changes to the Bondage menu.

INIT_ADB is an INIT to activate the 'interrupt' (Command-power) key on the IIsi and Mac LC machines. It is in the folder: '`Misc INITs & Source`'.

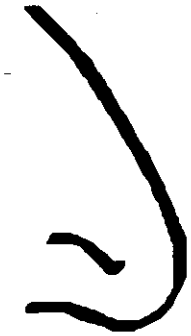
68040 machines and MMU Protection

The MMU in the 68040 is a simplified version of the MMU in the 68030 machines. I have spent some time studying the 040 MMU. At present I am of the opinion that it may be very difficult (or impossible) to implement MMU protection on 040 machines in a satisfactory way.
At this time I strongly suggest that you consider holding on to one of your 030 machines, preferably a IIfx in the case that you find the MMU protection feature useful, and I am not able to implement it for the 040 machines. I hope to have a resolution to this problem by November.

Installation Procedure

This update is shipped as a Self Extracting Archive (.sea) file. Double click on it to unpack the files.
NOTE Well - This version of the extractor program bombs randomly. If it does, then copy the .sea file to your hard disk, do a "Get Info" (Cmd-I) on it, and increase the Application Memory Size to a Meg or so, this should make it run properly.

Use the 'Install Debugger' script OR do the following:
Copy the files in the "put files in System fldr" to your System folder. **You must do this step.**
Copy or create a ROM .snt for your machine (default in Dbgr/Nosy files is for a 256K ROM Mac II)
Boot the mac, and validate that the new version is operational.
If you are using Version 3 of MacApp, then merge the IBS mods into the IBS folder.
If you are using IBS, then reinstall it, using the IBS_Install script.

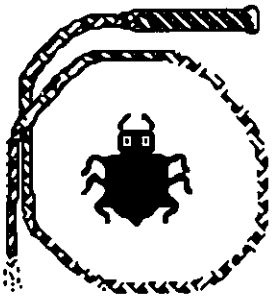


Information

Debugger Tech Note #9

The Debugger & MPW 3.2

July 91



Control

There are at least two of problems in MPW 3.2 final (ETO #4 - June 1991 ?) that will cause difficulties for MPW users who run The Debugger with MMU protection selected.

1) The 3.2 Final Shell has to be patched to avoid an MMU protection violation message from The Debugger. The ETO #4 release notes advises patching the Shell in CODE ,id=5 at offsets 46B4 & 4722,
changing $6704 -> $6004 in ResEdit.
This patch also applies to the version of the Shell distributed on the System 7 GM disk (May 1991).

As an alternative, If you have ETO #4 (June 1991), then I recommend that you install the MPW Shell from the QR or PQR folder. Both of these Shell's have been fixed so that they get along with The Debugger.

2) The library file (MPW:Libraries:Libraries:) Runtime.o contains procedures with data in the code segments that are modified by the procedure c2pStr. This will cause a MMU Protection violation error in Applications that you may build with the library.

This version of The Debugger distribution disk contains a revised version of the file Runtime.o **in the MPW Libraries (given to me by Apple) that will appear on the ETO #5 disk. The files are in the folder 'ETO #5 LIbraries'. They purport to fix the above problem. I personally have not had a chance to test them.**

3)The versions of Pascal 3.2 on ETO#2 & ETO #3 (and maybe ETO #4) have a bug in them that produce bad object files (.o) sometimes when -SYM ON is selected. I suggest that you stick with the 3.1 version of Pascal if you want to use the incremental compilation features of IBS. While I believe that it is possible to intermix object files from the 3.1 and 3.2 Pascal compilers, no one in Apple has been able to give me an affirmative answer to that question.

OBJECT MASTER

What is Object Master?

Object Master is a tool aimed at programmers using MPW (C, C++, and Pascal), THINK Object Pascal and C, Modula 2 - P1, and other programming languages on the Macintosh. Object Master greatly increases programmer productivity by providing intelligent source code editing, object browsing, resource browsing, segmentation mapping, and file mapping all in an environment fully integrated with the above programming languages via Apple Events.

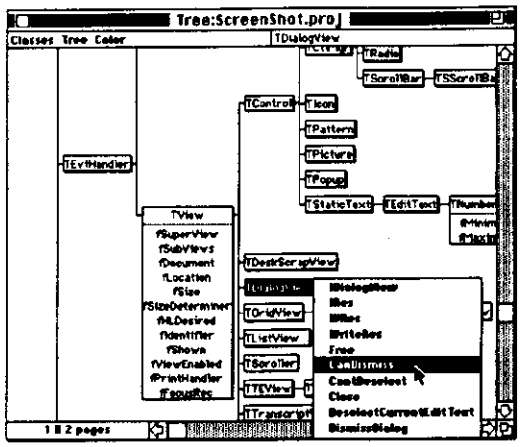
Intelligent Source Code Editing

One example of intelligent source code editing is the ability of the Object Master editor to do syntax checking on the fly, catching most errors without having to compile. As you create your files, a marker list is
automatically generated and provides easy access to all procedures and objects that have been defined in the file.

With Object Master, you can define custom code templates that are used to check the consistency of your personal coding style. The Object Master editor also allows you to assign unique colors and styles to procedures, keywords, and templates. Automatic indenting is also supported by Object Master.

Organize your files into a project

Object Master automatically maintains a database of data definitions and procedures. This organization of your files provides for fast navigation and structure editing. An Object Master project can mix Pascal, C, Modula, Rez, and 411 files. For example, a Pascal program that calls several C libraries can be edited as a single project with Object Master.



Class Tree View provides graphical display and navigation through the project's class hierarchy.

Features	Object Master[1]	MPW Editor	Think Pascal Editor	Think C Editor
Text Edit				
Styles for Keywords, Procedures/functions, Commands	Yes	No	Yes	No
Color for Keywords, Procedures/functions, Commands	Yes	No	No	No
Syntax Checking	Yes	No	No	No
Automatic Marker List	Yes	No	Yes	Yes[2]
411 Support	Yes	Yes	No	No
Generate Custom 411 Files	Yes	No	No	No
Make File Generation	Yes	Partial	n/a	n/a
Templates for User-Implemented Procedure Calls	Yes	No	No	No
Language Specific Macros	Yes	No	No	No
Search & Replace Text In Selection	Yes	No	No	No
Multi-File Search	Yes	Yes	Yes	Yes
Navigation				
Browsing Window	Yes	No	No	No
File Map	Yes	Yes	No	No
Graphical Class Tree	Yes	No	Yes	No
Segmentation Map (Procedure level)	Yes	Yes	No	No
Jump to Procedure/Class By Double-Clicking	Yes	None	Yes	?
Multiple Browse Windows	Yes	No	None	None
Multiple File	Yes	Yes	Yes	Yes
Resources				
Browsing	Yes	No	No	No

1. Features apply to all languages
2. With an INIT

Special EARLY USERS Version

OBJECT MASTER

Object Master is a tool aimed at programmers using MPW (C, C++, and Pascal), THINK Object Pascal and C, Modula 2 - P1, and other programming languages on the Macintosh. Object Master greatly increases programmer productivity by providing intelligent source code editing, object browsing, resource browsing, segmentation mapping, and file mapping all in an environment fully integrated with the above development environments via Apple Events.

Save $100 off the suggested retail price of $395.
Includes free upgrade to final version.

Offer expires ~~August 31~~ SEPT 30, 1991.

BILLING ADDRESS

Name _____

Address (No PO Boxes) _____

_____

City _____ State _____ Zip _____

SHIPPING ADDRESS

Name _____

Address (No PO Boxes) _____

_____

City _____ State _____ Zip _____

☐ I have read the license agreement on the reverse and agree to all its terms

Signature _____

ORDERING INFORMATION

Phone (required) _____

Unfortunately, ACIUS cannot accept CODs or Purchase Orders.

☐ MasterCard ☐ Visa ☐ Check Enclosed

Card Number _____

Expiration Date _____

Name on Card _____

Authorized Signature _____

	ObjectMaster	Sales Tax	Shipping	Total
☐ Non-California resident	$295.00	n/a	10.00	**305.00**
☐ California resident	295.00	24.34	10.00	**329.34**

Mail, Phone, or FAX your order to ACIUS:

ACIUS, Inc.
10351 Bubb Road
Cupertino, CA 95014
(408) 252-4444
FAX (408) 252-0831
AppleLink D4444

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

March 25, 1991

Dear Mac II Nosy/Debugger User,

Enclosed is the latest release of the Universal Version of *MacNosy* and *The Debugger V2/91*, and some accompanying notes on them. This release contains numerous bug fixes and new features which are summarized in the Change History file on the update disk.

For those of you who purchased *The Debugger* prior to January 1, 1991, this is you last free update. To receive future updates, please fill out the enclosed invoice and return it to me (before May 1, 1991 please) with a check for $130 (payable to a US Bank) or authorization to bill your credit card. Those of you who purchased *The Debugger* after 1/1/91 and those of you who "re-up" will receive updates and support in 1991.

Site Licenses

Site (multiple copy) Licenses are available through Jasik Designs. The pricing is $225 per copy times the number of copies (programmers using the product).

Real Memory, Virtual Memory & Maxima

As part of the co-marketing agreement with Connectix, you may purchase **Virtual** or **Maxima**. Given the price of 1 meg SIMMs ($40), it is my recommendation that you fill up with 8 Meg first. Under system 6.0.x, Maxima enables one to:
1) Use some part of memory as a RAM disk that is preserved across "warm boots".
2) take advantage of 4 Meg SIMMs by using the memory over 8 Meg for Applications and/or RAM disk
Be aware that Dove, MicroTech and Newer are **bundling Maxima with 4 Meg SIMMs.**
For more info on Maxima, Virtual, you may call Connectix at 800-950-5880.

Tech Support is available via phone, Applelink (D1037) or CompuServe.
Apple is in the process of starting a Debugger Support Area on AppleLink, look for it as I will use it as a focal point for announcements, etc.

Installation Procedure

This update is shipped as a "Compact Pro" Self Extracting Archive file. Copy the file to your hard disk, and double click on it to unpack the files. Use the 'Install Debugger' script OR do the following:
Copy the files in the "put files in System fldr" to your System folder. **You must do this step.**
Copy or create a ROM .snt for your machine (default in Dbgr/Nosy files is for a 256K ROM Mac II)
Boot the mac, and validate that the new version is operational.
If you are using IBS, then reinstall it, using the IBS_Install script.

System Compatibility - Recommended System Level

This version of The Debugger and MacNosy has been tested on System 6.0.5, 6.0.7, 7.0ß4 & 7.0ß5.
I have had minimal experience with the new machines (Classic, LC and IIsi), for them you must build a ROM/CODE.snt file from scratch.

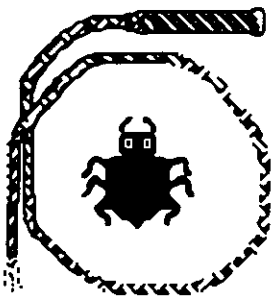
New Features, etc

- Works better with C++ (correct display of structures derived from PascalObject, etc)
- IBS fixes to take advantage of the partial compilation in Pascal V3.2 on ETO #3.
- The Debugger is now operational in 32 bit mode on System 7 (tested on a IIfx with 20Meg).
- Operates with AUX 2.0.1 in 24 and possibly 32 bit mode (leaves dead cursors on the screen).
- Kludges have been worked out so that one can use the 8*24 GC card with System 6.0.x.
- Supports some 68040 Accelerator boards (TokaMac, Radius Rocket).
- Some new debugger flags have been added (MB_DX, Del_Rsrcs, Ignore_NIL).
- For C++ programs & Cmd-D - if a partial name (Class::mmm) is typed in, then open up the procedure that contains that name OR list all procedure names that 'match' in a window.

For example, if `Txyz` is a class, then entering "`txyz::`" into the Cmd-D box will list all its methods.

Debugger Tech Note #8

The Debugger & System 7, etc



March 91

Information

Control

The Debugger vs System 7

For me, System 7 has been a royal pain in the butt that has forced me to direct my energies towards making basic changes in the low level operation of The Debugger, instead of adding useful external features and fixing bugs in existing ones. I expect to spend some more time this spring incorporating various parts of the ROM into The Debugger so it is independent of the Layer Manager, etc. For the moment, I have made my peace with it by unpatching most of their traps in The Debugger's world, but I do not see this as a path for long term survival of the product, particularly on the older macs, such as the II, IIx and IIcx.

Another thing that you should be aware of is that the Finder and most of the resources in the System file are packed (encrypted). In order to disassemble them you will have to use ResEdit to unpack them, by copying them off to another file. Nosy does not use the Resource manager to read resources, and does not presently know how to unpack them. Resources are unpacked by the entry point jCheckLoad. MultiFinder, DA Handler and Backgrounder are hiding in the System file as resources of type 'scod' with weird id's.

A few of you have run into the problem on System 7, that when xDbgr_Startup asked you to "Locate The Debugger", it bombed instead of bringing up the Pack3 (SFGetFile) dialog. To get around this problem I have added a little Application; Set_Dbgr_Dir; which will set the DirID of the Debugger's folder in xDbgr_Startup. You bypass Launching The Debugger, and then use Set_Dbgr_Dir to "Locate The Debugger". Then you may boot and Launch The Debugger.

For those of you who do make it to the Pack3 dialog in xDbgr_Startup, use the Desktop button to go up, DO NOT try to use the pop up menu to navigate your way to the folder that The Debugger is in.

New Debugger Flags

I have added a couple of new flags to The Debugger, they are:

Hex_Base =1 to display Longs as Hex in the Values and Local Values windows when a .SYM file used.

MB_DX - similar to the DX flag in Macsbug for those of you who litter your program with _DebugStr calls.

Ignore_NIL - If set to 1, then Discipline will not report NIL Handle/Ptr violations from DisposHandle, etc.

Del_Rsrcs - When set to 1, The Debugger will delete 'tasks' when _CloseResFile is called to close the resource file associated with the task. The Flag is set to 0 in the ROM.dsi so that tasks associated with INITs persist until the next boot, and is automatically set to 1 on Finder launch. You may set/clear it after that as you wish.

ToolServer & The Debugger

The Debugger has been fixed so that it can run with ToolServer, the background Shell. In order to avoid "IsDocTask" failed messages remember that you have run `InformDbgr` in both the MPW Shell and ToolServer.

Operation of The Debugger with the 8•24 GC card & Acceleration ON

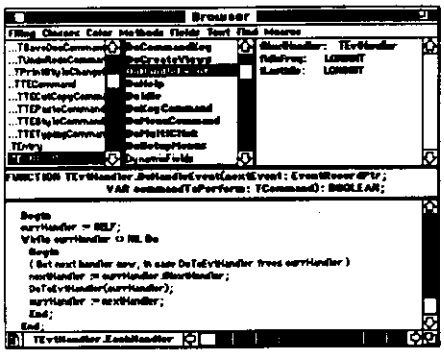
The Debugger does not get along very well with the 8•24 GC card when acceleration is turned ON.
If you have to operate your system in this mode with The Debugger, then the following must be true:

1) you MUST have more than one monitor.
   if you are running System 7.0ßx, then ignore the rest, as there does NOT appear to be a problem in this case.
   You may have to select Split Screen mode in The Debugger.
3) If you are running 6.0.5 or 6.0.7 then you must do the following:
   Change the name of "xDbgr_Startup" to "Dbgr_Boot" so that The Debugger loads at "Macsbug"time
   You may debug INITs that load after "Dbgr_Boot".
   You must select "Split Screen" mode.

OBJECT MASTER

Browser Windows

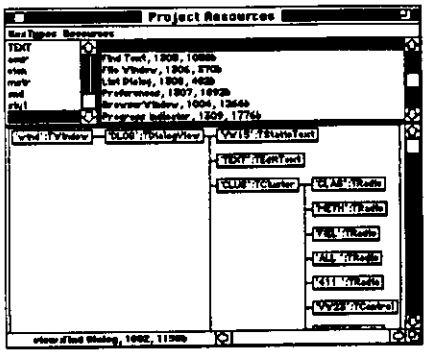
Class hierarchies can be viewed in either outline form or via a graphical hierarchy. You can view classes details by double clicking on a class, revealing its header information, methods, field types, and inheritance information. All editing functions can be performed from the edit panel within a browser window. The number of browser windows that can be open is limited only by the amount of available memory.



The Browse Window combines easy navigation with full source code editing facilities. You can open as many Browse Windows as memory allows.

Segmentation, File, and Resource Mapping

Segmentation and File lists provide a quick way of viewing the segmentation plan for a given project. You can view resources from within Object Master. If you wish to edit them, double clicking on one will launch ResEdit and open the appropriate resource.You can customize your resource view by resource type allowing t you to work with resources in the way best suited to your application.



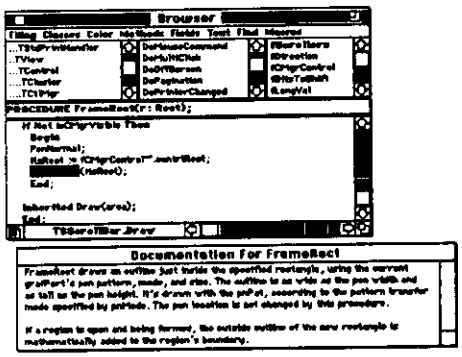
Object Master gives the programmer customizable resource previews specifically designed for programming reference. ResEdit (or other resource editors) can be automatically opened for editing directly from the Resource Preview.

Interactive Compiling

Projects can be compiled in the background with the MPW tool server. A list of errors is returned to Object Master and displayed in a scrolling window. When you click on an error, Object Master takes you to the line of code that generated the error.

On-Line Documentation

Access is provided on-line to 411 (Inside Macintosh) documentation. You can also automatically generate your own 411 (help) file from your source code.



Object Master provides full access to all 411 documentation. In addition to using the 411 files provided by Apple, Object Master can automatically create 411 from your own source code.

Object Master was written by Loïc Vandereyken

Worldwide distribution:
ACI, Paris, France: 1-42-27-37-25

Distribution in the United States:
ACIUS, Cupertino, CA: (408) 253-3366

ACIUS, Inc.
10351 Bubb Road
Cupertino, CA 95014
(408) 252-4444

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

October 31, 1990

Dear Mac II Nosy/Debugger User,

Enclosed is the latest release of the Universal Version of *MacNosy* and *The Debugger V2/90*, and some accompanying notes on them. This release contains numerous bug fixes and new features which are summarized in (revised) Debugger Tech Notes 6 and 7, and the (unexpurgated) Change History file on the update disk. The major feature additions of this release are compatibility with the System 7.0ß1 release, and the use of the MMU to protect the Macintosh from the program being debugged.

Tech Support

Available via phone, Applelink, CompuServe, jungle drums, house calls, etc.

Installation Procedure (unpack and add water)

Update your Debugger Tech Notes by removing the old 6 & 7 and replacing them with the new ones.

This update is shipped in "Auto UnStuffit" format.

Create a folder on your hard disk to deposit the unstuffed files in. Copy the Auto Unstuffit file to that folder and double click on the Auto UnStuffit Icon to unstuff the files.

Complete (add water) the "Dbgr/Nosy files" folder by:

a) copying in your existing ROM/CODE.snt file from the old "Dbgr/Nosy files" folder

b) if you have a 512K ROM Mac (IIci/IIfx/IIsi) use the one in the "Mac IIci/IIfx ROM .snt" folder.

c) or create one in Nosy

Copy the files in the "put files in System fldr" to your System folder. **You must do this step.**

You can trash your Finder and MultiFinder .snt files (in the System folder) as they are now optional.

Boot the mac, and validate that the new version is operational.

While I have included a complete IBS folder on the disk, the only file in IBS that has changed is PatchLink (':IBS V1 for MPW 3.x:IBS *f*:Tools:PatchLink'). Copy it to the MPW Tools folder.

MacApp users: make sure to pick up the latest mods to `UObject.globals.p`

I made some changes in August & October to counteract the fact that the release MacApp 2.0 no longer sets the low memory globals. I revised the code in `InitUObject` to set them only if The Debugger is present and the MacApp application is being debugged. With this fix in, one will be able to run more than one MacApp based application at a time.

General Comments

Part of the reason that this release is over a month late was that I did fairly extensive testing of the MMU protection code on a variety of configurations, and testing of The Debugger on System 7.

I was unable to get The Debugger to work with System 7 VM, and haven't even tried to get it to work in 32-bit mode (I am well aware that The Debugger is only 24 bit clean).

As mentioned in the last release, the Shutdown Debugger command (Files Menu) was removed.

System Compatibility

This version of The Debugger and MacNosy has been tested on System 6.0.4, 6.0.5, 6.0.7 and 7.0ß1. It is independent of the version of TextEdit, and should run somewhat better with the Kanji systems. If you are using any of the Kanji systems, you should set the flag `BAD_Zero` to 0 in the ROM.dsi file.

I have had minimal exposure to the new machines (Classic, LC and IIsi). For the IIsi, you should be able to use the IIci/IIfx ROM/CODE.snt. For both the Classic and the LC you should build a ROM/CODE.snt from scratch.

C++ and other crimes against nature

I would rather be stranded on a desert island with Rosanne Barr, than use C++, but many of you are giving it a whirl. Note that if you mix C++ and MacApp with PascalObjects, then you must make your object names uppercase if you want to see them in my object inspector. The problem is that I made The Debugger sensitive to the case of types to handle C and C++ types properly, but somewhere along the line the compiler (c++ ?) or the Linker uppercases them in the MacApp method table.

SOFTWARE LICENSE AGREEMENT

"Licensor" [ACI or, in the United States, ACIUS] grants you ["Licensee"] a non-transferable, non-exclusive license to use this copy of the program and accompanying materials according to the following terms:

LICENSE:

You may:

a) use the program on only one computer at a time;

b) make one (1) copy of the program in machine readable form solely for backup purposes, provided that you reproduce all proprietary notices on the copy;

c) physically transfer the program from one computer to another, provided that the program is used on only one computer at a time; and

d) transfer the program onto a hard disk only for use as described above provided that you can immediately prove ownership of the original diskettes.

You may not:

a) use the program in a network unless you pay for a separate license for each terminal or workstation from which the program will be accessed;

b) modify, translate, reverse engineer, decompile, disassemble, create derivative works based on, or copy (except for the backup copy) the program or accompanying materials;

c) rent, transfer or grant any rights in the program in any form or accompanying materials to any person without the prior written consent of Licensor which, if given, is subject to the conferee's consent to the terms and conditions of this license; or

d) remove any proprietary notices, labels or marks on the program and accompanying materials.

This license is not a sale. Title and copyrights to the program, accompanying materials and any copy made by you remain with Licensor.

TERMINATION

Unauthorized copying of the program (alone or merged with other software) or the accompanying materials, or failure to comply with the above restrictions will result in automatic termination of this license and will make available to Licensor other legal remedies. Upon termination you will destroy or return to Licensor the program, accompanying materials and any copies.

LIMITED WARRANTY AND DISCLAIMER

THE PROGRAM AND ACCOMPANYING MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Licensor does not warrant that the functions contained in the program will meet your requirements or that the operation will be uninterrupted or error free. The entire risk as to the use, quality, and performance of the program is with you. Should the program prove defective, you, and not Licensor, assume the entire cost of any necessary repair.

However, Licensor warrants the diskettes on which the program is furnished to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt. The duration of any implied warranties on the diskettes is limited to the period stated above. Licensor's entire liability and your exclusive remedy as to the diskettes (which is subject to you returning the diskettes to Licensor or an authorized dealer with a copy of your receipt) will be the replacement of the diskettes or, if Licensor or the dealer is unable to deliver a replacement diskette, the refund of the purchase price and termination of this Agreement.

SOME STATES DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

LIMITATION OF LIABILITY

IN NO EVENT WILL LICENSOR BE LIABLE FOR ANY DAMAGES, INCLUDING LOSS OF DATA, LOST PROFITS, COST OF COVER OR OTHER SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES ARISING FROM THE USE OF THE PROGRAM OR ACCOMPANYING MATERIALS, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY. THIS LIMITATION WILL APPLY EVEN IF LICENSOR OR AUTHORIZED DEALER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. YOU ACKNOWLEDGE THAT THE LICENSE FEE REFLECTS THIS ALLOCATION OF RISK. SOME STATES DO NOT ALLOW LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

GENERAL

This Agreement will be governed by the laws of France [in the United States, the laws of the State of California]. In any dispute arising out of this Agreement, Licensor and you each consent to the jurisdiction of the courts of France [in the United States, both the state and federal courts of Santa Clara county, California].

Use, duplication or disclosure by the U.S. Government is subject to restrictions stated in paragraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013.

Licensor: ACI, 5 Rue Beaujon, 75008 Paris, France [in the United States: ACIUS, 10351 Bubb Rd., Cupertino, CA 95014]

This Agreement is the entire agreement between us and supersedes any other communications with respect to the program and accompanying materials.

If any provision of this Agreement is held to be unenforceable, the remainder of this agreement shall continue in full force and effect.

If you have any questions, please contact: ACI Customer Service, (33) 1 42 27 37 25 [or in the United States: ACIUS Customer Service, (408) 252-4444]

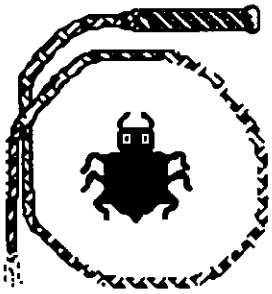
SIGN AND MAIL THE REGISTRATION CARD TODAY.

Return of the registration card is required to receive any product updates and notices of new versions or enhancements.

Debugger Tech Note #6

Change History - 3/89 to 10/90

Oct 90



Information

Control

A **summary of external changes** to The Debugger that were made after version 2 of the manual was printed. Changes discussed in other Debugger Tech Notes are not mentioned here.

- Handle C++ programs, "demangling" the names & handling names of the form "class::method".
  Current release demangles the constructor/destructor to "class::__ct" and "class::__dt".
  There are no special provisions for overloaded names.
- Add source level stepping for programs with .SYM files.
- Change single screen mode to Swap Screens versus Split Screen modes.
- Extend the type compiler to handle enumerated types, integer subranges, Pascal OBJECTs,etc.
  See the file "AppleTalk_EQUs" for examples.
  typName = OBJECT(className) fieldList Methods END; is the syntax for Objects.
- Allow the redefinition of types introduced in a .SYM file via **=TYPE,redef** in the ".dsi" file. (DTN #1)
  Redefine your types to improve the display of them when a compiler "mangles" their definitions.
- Allow variable upper bound in last array of a record definition (Do a Cmd-space of TeRecord, etc).
- Fixed is now a base type that displays a longint as a flting pt number (16/integer_part,16/fraction)
  Expressions Parser - integers appearing in type declarations default to decimal base unless explicitly preceded by a '$". The "$" is NOT persistent as it is in arithmetic expressions.
- Record display - Add a kludge to auto recognize Color Grafports and new style TeRecords.
- Add "Dbg_Rsrcs" flag in .dsi files to turn off the recording/debugging of resources.
  This was done so as to minimize screen flashing when the user has only one screen.
- Add the "flag" BAD_ZERO to control the "garbaging" of Loc zero.
- Cmd-W of a local var name will bring up the "Local Vars-" window & highlight the name.
- Allow local and global variable Value windows to be dynamic (**Cmd-8**).
- **Changes to Both MacNosy & The Debugger**
- Work with both MPW 3.0 , 3.1 and 3.2 SYM file formats
- Cmd-shift-Minus of a decimal number will convert it to Hexadecimal
- Add AND, OR, NOT and DIV to the operators in the expression parser.
- Cmd-period to cancel a dialog or to abort reading of the .SYM file, etc.

"Source Level Debugging" - Think's LightSpeedC Projects

For LightSpeedC projects compiled with **"Use Debugger" on**, *The Debugger* will display the source code of the procedures in windows. One may optionally toggle between source and interspersed ASM with a Command-Click in the content region of the window. The types of the global variables may be supplied by the user in a ".dsi" file. The names of the local variables may be inferred from the ASM display.

Oops, The following didn't make it into the Manual:
To pass parameters to a "?UserP", use assignments to global variables in the "action clause" that calls it.

The Scroll bar in the "**Type@nnn**" windows is a "page" type scroll bar that does a page up/down when clicked in the gray region. It is useful in the case you have an array or record. An example is to define the type:
`trapTbl = record a,b,c,d:Array[0..3] of ProcPtr end;` and do a `"trapTbl@400"`.

Known Problems

- Overall, this release is a bit on the spotty side, in that there are number of areas where the features are not complete, or have restrictions that they shouldn't have, etc.
- The MMU Memory Protection does not work correctly if you are **using internal Video (IIci, IIsi),** nor does it work with the DOVE 030, and it also appears to have problems with TokenTalk, A/Rose. I will be looking further into fixing those problems in the the next release.
- Operation of The Debugger when FileShare is active is unknown.
- The Debugger does not handle .SYM file info for arrays with variable bounds.emitted by LS Fortran.

Future (Work in Progress)

We will be working on the following features for the next release (late May or June 1991):

- Fix problems with the MMU Memory Protection code described above.
- Support for the Model Far option in MPW 3.2
- Full compatibility with System 7, addition of trap calls and types, operation with VM, etc.
- Miscellaneous enhancements to trap discipline
- Enhance expression parser to allow writes to alternate windows, loops, etc.

Debugger Tech Notes (DTN's) on Disk

The Debugger Tech Notes are now distributed on disk in SuperGlue Viewer format, you may view them from AppleLink or SuperGlue.

The notorious ROM.snt message (excuses, excuses)

When The Debugger starts up, the '-Notes-' window contains the following lines:

```
reading ROM/CODE.snt
System PTCHs are different, you may wish to explore ROM
• addresses of ROM patches will NOT be adjusted
```

You should ignore the message. Both Nosy & The Debugger contain code that attempts to adjust the addresses of the RAM patches to the ROM so that they are correct relative to the current set of patches/INITs, etc.
In most cases the algorithm is failing, and the last two lines are the result. The net effect of the failure is that if you attempt to display a RAM patch in The Debugger via Cmd-D that you may get a garbage window.

Monitors and Video Cards that cause problems

A variety of monitors, and video cards that try to do more than their assigned task, cause problems to The Debugger when one uses certain commands in it. An example is trying to use the Trap Entry Trace command with the E-Machines Z-21 monitor. One ends up executing the action clause only for the _SwapMMUMode calls executed by the video driver. In general, the fancier the features of the monitor, the more likely that it and its associated software will interfere with the operation of The Debugger. Other hardware that interferes with The Debugger is Apple's 8•24 GC card, SuperMac monitors with pan and Zoom, and I suspect the Radius Pivot..

Shows I will Attend

I will be at the Apple Developers conference in May, MacHack in Ann Arbor in June and the Boston Macworld in August (exhibiting at the Apple Tools expo). I will give out updates on an exchange basis, so bring your labeled Debugger disk with you.

The Head Nose on Tape

I hope to spend some time this summer creating an instructional video tape on The Debugger & MacNosy. It will be in VHS format (NTSC), and if I do carry this promise (threat) through, then I will be sending copies out to you in the summer/fall.

Sincerely,

Steve Jasik **AppleLink: D1037 Compuserve ID: 76004,2067**

Change History from 3/8/90 to 10/17/90

MacApp 2.0 change files
- put the defines of the low mem cells back into Uobject.Globals.p so current versions of The Debugger work.
- incorporate 2.0p2 code into change files
- add code to avoid Bus Errors when modifying a CODE segment

IBS
- Add patch file for 3.2ß1 Shell, and notes about upcoming '`duplicate -o`'
- NEED mods for 3.2 Pascal, for present forgo partial compile or use 3.1 Pascal

Both Nosy & Debugger
- ESC key clears any hiliting in front window, VERY useful
- Make processing of types from SYM file case sensitive if it has type names with lower case chars in it.

Nosy
- Restore Save .map file option
- fix - Long standing bug that caused blowups when listing long lines to console in TTY mode

Debugger - enhancements & fixes
- ADD MMU memory protection (see DTN #7 for details)
- Sharpen checking for bad data addresses so don't blow up in The Debugger
- Remove ShutDown Debugger command & _DisposHandle fstFree Check
- handling of INITs so that task persists after INIT finishes
  Now one can symbolically debug the patches that an INIT makes
- Add error message to tell user name of Void/Bad types when 1st read of .SYM file.
- allow "addr:value" syntax in SET... command
- improper patch to _SetTrapAddr caused blowups in 3.2ß1 Shell
- fix to handle new _SysError error numbers properly, -490, -491, etc
  Add Dbgr_StoreWord & Dbgr_StoreLong Procedures to allow storing into CODE segments without causing a memory protection violation
- Flush the Instruction cache to make Conditional Bkpt's work reliably
- Add kludges to do MMU Protection on IIfx's with more than 8 Meg of RAM
- Proc Entry/Exit tracing - allow condNum,TskNum to restrict tracing to only calls from the specified task
- fix - Dbgr was not evaluating conditional expressions correctly (bug introduced in 3/8/90 Debugger)
- Reduce xxx.SYM is out of date Alert to msg in Notes & Beep
- Fix and enhance Trap Discipline, add additional checks. It will now be active at Boot time, if selected.
- Fix Handle Zapping & Discipline to work with Maxima
- Fix the Rsrc map & hz@ windows so they can update dynamically
- Allow user to shutdown Finder in MF mode
- Delete Pack2 from Debugger as it was never being called

The Handling of Types (both Nosy & The Debugger)

Formerly The Debugger handled type names in a case insensitive manner. For example the type names "TLIST" and "TList" were recognized as the same type. Due to the fact that C and C++ are are case sensitive languages, I changed the name search/entry routines to also be case sensitive.

When a MPW .SYM file is initially processed, the user types are entered and converted into an internal format unique to The Debugger. Type names that correspond to standard Macintosh types (GrafPort, etc) are mapped into the built in type definitions. If any of the user type names contain a lower case letter, then the language is assumed to be one in which type definitions are case sensitive. This fact will be reflected in the menu bar by changing the minus in the task menu (nn - taskname) to a plus.

All this was done so that C and MacApp/C++ programs would work properly.

NOT Yet Done as of 10/17/90

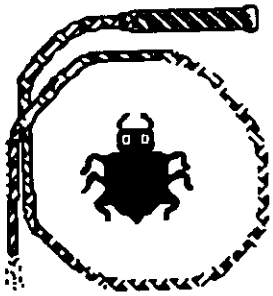
Make Debugger operate with System 7 VM and Fileshare
Mods to IBS for MPW 3.2 Pascal, etc
Fix problems with Nosy ROM explore of Mac LC
Get the right sound resources in The Debugger for System 6.0,6 & System 7, etc

Information

Debugger Tech Note #7

MMU Memory Protection

October 90



Control

The Goals of MMU for Memory Protection are:
- To help the programmer find bugs when they occur, not millions of instructions later.
- To protect the rest of the Macintosh address space from the program being debugged
- To protect the program being debugged from writing into its CODE resources.
- To accomplish the above with a minimal slowdown of the system.
- To be reasonably "transparent" to use.

The Limitations are:
- Requires a Mac II with a MC68851 PMMU or a later Mac with a MC 68030 CPU.
- As there is only one master of the MMU, you may not use it with any form of Virtual Memory which includes Virtual, Maxima, System 7 VM, etc.
- Your machine should have 4 - 8 Meg of RAM. IIfx's with more than 8 Meg will run as if they have 8 Meg when MMU protection is selected (at Debugger startup time).
- When Internal Video is being used, the screen will be swapped in Step Continuous mode.
- MMU protection only applies to Applications running in their own heap (no embedded processes such as MPW Tools), and only to 1 program at a time.
- There is no special support for the use of MultiFinder Temp Memory.
- When another process such as the Finder or the MPW Shell is running, only The Debugger is protected.
- The MMU Memory protection code went through an extensive (and exhausting) beta test cycle. Despite that there are some hardware configurations on which it may not run.

The Model is:
In a mainframe a task operates within an assigned memory space and makes system calls in order to communicate with the outside world. This is the model I adopted for MMU Memory Protection. The memory space of your Application starts with the Heap Zone header and ends with the last byte of the (CODE 0) jump table that is above A5. The System occupies the rest of the address space (ROM, System Heap, etc), and its only restriction is that it may not write into The Debugger's space.

The implications are:
In addition to meeting the above stated goals, your Application may not write to (most) low memory globals directly, nor write directly to the screen or I/O devices, so you will have to move custom MBDFs, CDEFs, etc into the System Heap if you wish that they run without getting spurious MMU Protection Violation messages.

General
While the combination of MMU Memory Protection and Trap Discipline can help programmers find certain classes of errors at the point where they are committed, they are not an absolute security blanket, and there are a wide variety of stupidities that that one can commit that cannot be protected against by the most advanced debugger. The moral is: Programming is an art, keep you wits about you, stay on guard, and occasionally remember that compiling your program with array bounds range checking may help you discover errors.

The Debugger vs System 7

For me, System 7 is a royal pain in the butt that has forced me to direct my energies towards making basic changes in the low level operation of The Debugger, instead of adding useful external features and fixing bugs in existing ones. I expect to spend some more time this winter incorporating various parts of the ROM into The Debugger so it is independent of the Layer Manager, etc.

For the moment, I have made my peace with it by unpatching most of their traps in The Debugger's world, but I do not see this as a path for long term survival of the product, particularly on the older macs, such as the II, IIx and IIcx.

Another thing that you should be aware of is that the Finder and most of the resources in the System file are packed (encrypted). In order to disassemble them you will have to use ResEdit to unpack them, by copying them off to another file. Nosy does not use the Resource manager to read resources, and does not presently know how to unpack them. Resources are unpacked by the entry point jCheckLoad. MultiFinder, DA Handler and Backgrounder are hiding in the System file as resources of type 'scod' with weird id's.

A few of you have run into the problem on System 7, that when xDbgr_Startup asked you to "Locate The Debugger", it bombed instead of bringing up the Pack3 (SFGetFile) dialog. I have not been able to track this problem down, but here is a workaround. Both xDbgr_Startup and RunDbgr contain a resource of type TDid which holds the DirID (Directory ID) of the folder that The Debugger is in.

You can set the value of it, by booting with System 6, and copying that copy of xDbgr_Startup into the System 7 Extensions folder.

For those of you who do make it to the Pack3 dialog, use the Desktop button to go up, DO NOT try to use the pop up menu to navigate your way to the folder that The Debugger is in.

Known Problems

- Transient resources such as Boomerang cause The Debugger's internal task table to get fouled up, if the Dbg_Rsrcs flag is on. Use Dbg_Rsrcs only as necessary until the problem is fixed.
- When doing the ROM explore on the Mac LC, Nosy appears to get lost for a while, but recovers, and build the ROM .snt OK.
- MMU Protection for MPW Tools did not make it.
- Debugging of HyperCard XCMDs does not work with Hypercard 2.0 as it calls _LoadResource which The Debugger does not patch. Use Hypercard 1.2.x if you have to debug XCMDs.
- The Debugger does not handle .SYM file info for arrays with variable bounds.emitted by LS Fortran.
- Some programs do not like to run above 8 Meg, and cause strange blowups in The Debugger.
- The MMU Swap Screen option does not work on a Mac IIci if you have less than 8 Meg of RAM.

Future (Work in Progress)

We will be working on the following features for the next release (March 1991):

- Full compatibility with System 7, addition of trap calls and types, etc. (see below).
- Addition of a high speed performance analyzer, and "coverage" analyzer to tell you what parts of your program you did or didn't execute (useful for SQA types).
- Enhance expression parser to allow writes to alternate windows, loops, etc.

The Head Nose on Tape

I hope to spend some time this fall creating an instructional video tape on The Debugger & MacNosy. It will be in VHS format (NTSC), and if I do carry this promise (threat) through, then I will be sending copies out to you in the spring.

If you have problems with this release, give me a call. Feedback improves the breed.

Sincerely,

Steve Jasik **Compuserve ID: 76004,2067 Delphi: "MacNosy" AppleLink: D1037**

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

March 10, 1990

Dear Mac II Nosy/Debugger User,

Enclosed is the latest release of the Universal Version of *MacNosy* and *The Debugger V2/90*, and some accompanying notes on them. This release contains numerous bug fixes and new features which are summarized in Debugger Tech Note #7, and the Change History file on the update disk. The major feature additions of this release are enhancements to the screen handling for 32-bit base address monitors , the use of the MMU to avoid screen swapping when using the animated trace (step continuous) and compatibility with the Virtual and Maxima INITs from Connectix.

For those of you who purchased *The Debugger* prior to September 1, 1989, this is you last free update. To receive future updates, please fill out the enclosed invoice and return it to me with a check for $105 (payable to a US Bank) before May 1, 1990. Those of you who purchased *The Debugger* after 9/1/89 and those of you who "re-up" will receive updates and support in 1990.
As part of a co-marketing agreement with Connectix (Virtual INIT), you may pay your update fee with your Visa or MasterCard, and optionally buy a copy of "Virtual" and/or a MC68851 PMMU if you have a MacII without one. I strongly recommend the purchase of PMMU if you do not have one.
There will be a late charge of $25 for update fees that are received after May 1, 1990.

Site Licenses

Site (multiple copy) Licenses are available through Jasik Designs. The pricing is $225 per copy times the number of copies (programmers using the product).

Real Memory, Virtual Memory & Maxima

As part of the co-marketing agreement with Connectix, you may purchase **Virtual** or **Maxima**. Given the price of 1 meg SIMMs ($63), it is my recommendation that you fill up with 8 Meg first.
Maxima enables one to:
1) Use some part of memory as a RAM disk that is preserved across "warm boots".
2) take advantage of 4 Meg SIMMs by using the memory over 8 Meg for Applications and/or RAM disk
Be aware that Dove, MicroTech and Newer are **bundling Maxima with 4 Meg SIMMs.**

Tech Support

Available via phone, Applelink or CompuServe. I am phasing out the Delphi group.

Installation Procedure

This update is shipped in "Auto UnStuffit" format. Create a folder on your hard disk to deposit the unstuffed files in. Copy the Auto Unstuffit file to that folder and double click on the Auto UnStuffit Icon to unstuff the files.
There is not enough space on the floppy to hold two ROM .snt files, so complete the "Dbgr/Nosy files" folder by copying in your existing ROM/CODE.snt file from the old "Dbgr/Nosy files" folder if you have a 256K ROM Mac I. If you have a Mac IIci or other 512K ROM Mac, then copy the ROM.snt file from the "Mac IIci ROM .snt" folder.
Copy the files in the "put files in System fldr" to your System folder. **You must do this step.**
You can trash your Finder and MultiFinder .snt files as they are now optional.
Boot the mac, and validate that the new version is operational.
If you are using IBS, then reinstall it, using the IBS_Install script.

System Compatibility - Recommended System Level

This version of The Debugger and MacNosy has been tested on System 6.0.3, 6.0.4, beta versions of 6.0.5, and alpha versions of 7. If you have not updated to System 6.0.4 for development purposes, please do so in the near future. Use of the MMU Swap Screen option requires System 6.0.4 or later.

Change History from 9/89 to 3/90

Changes to The Debugger
- Finder and MultiFinder .snt files **are now optional**
- The Debugger now works with the Virtual and Maxima INITs (from Connectix)
- Allow $Annn syntax in Cmd-J to set trap intercepts
- Hiliting a Dbgr flag name and Ω-F Toggles the value of the flag
- Add flag Dbg_ThC to control the auto Debugging of Think C projects
- Add display of Control & MMU registers for 680x0 - (`Aux_Reg@nnn` in the -Registers- window)
- Fix - Debugger can now handle programs with up to 254 CODE segments
- Fix - always patch __setjmp so that one can Step thru it (MPW C programs only)
- write(var) where var is a string, array or record displays the object in its proper format
- Fix - allow both string; and string[n]; in type declarations
- Add - ?Rec(@GlobalVar,addr) where GlobalVar:String; holds the name of the type
- Fix - WatchPoint - allow fba to be any address
- Add support for **LS Fortran V2** common variables and Arrays (needs MPW 3.1 or later Linker).

Nosy changes
- demangle C++ Macsbug names & recognize Macsbug names in ROM patch area
- if file name contains the string "Macsbug" then disassemble the Data fork

IBS Changes
- Add an Interfile goto marker facility, and scripts to set it up (see Tech Note #4)
- fixes to PatchLink to process MacApp names properly,
  correctly process multiple changes in multiple files, etc
- Add Object list collector to MacApp, Object Inspector Menu to Dbgr, etc (see Tech Note #2)
- Add patch files for MPW 3.1 final, and make changes to IBS_Install script.
- Change MABuild mods to auto create the IBS project, and touch the .SYM file

General Changes to Both Nosy & The Debugger

- Fix to work with IIci, Portable, IIfx & System 7
- Revise Type processing so we can handle upto 8192 user types in a .SYM file
- Fix bugs associated with processing files on multiple volumes
- Add - select Line when triple Click is detected
- Do balanced match hiliting for ({ [' " ` and / (a la MPW Shell)

The behavior of **Cmd-G** was changed as follows:
For Cmd-G the search is case insensitive and ignores token boundaries.
For Cmd-shift-G the search is case sensitive and the search object must be a token.

Note that the default **Cmd-F** search when an item is highlighted in the front window is case sensitive and the search object must be a token.

The Behavior of **Cmd-U** was changed as follows:
Cmd-U (Update Globals) and Cmd-shift-U (update Local vars) have been combined into one menu item
A new menu item "Misc:Locals in Current proc" (Ω-U) has been added.

The behavior of the write statement (Debugger Manual page 38) was changed so that write(var) where var is of type string, Cstring, record or array will print out the the contents of the object in the expected format. Note that you can force a variable or expression to a type in a write statement by using the notation write(VarName:type).

Nosy and mis-segmented procedures
To view a list of the procedures that are referenced only once and in a different segment than the caller do the following: After a Treewalk, press Cmd-Y to enter TTY mode, then press M and return to go to the Maps menu. Enter "oxx" and return to open a file. Enter "s1" and return.
When the list is finished enter "o" return, return, "d" return. Then open the file "xx".
MacApp users - The map will contain spurious entries for method names, ignore them.

MPW C Programs - Important Note
The Debugger automatically translates the type **Array of Unsigned char** into type string, and displays variables as such. **Arrays of Char** translate to type SignedByte on the Mac are NOT given the same special treatment.

Order Form

March 8, 1990

From: Jasik Designs, 343 Trenton Way, Menlo Park, CA 94025

To; mailing label New Address _____
_____
_____

For:

Quantity	Item	Unit price	Total
_____	Additional Licenses for Univ Debugger	**$225**	_____
_____	**Maxima** memory enhancer **&** RAMdisk	**$75**	_____
_____	**Virtual** 030 for Mac IIx, IIcx, IIci & SE/30	**$99**	_____
_____	**Virtual** for Mac II (Software **and PMMU**)	**$175**	_____
_____	**Virtual** for Mac II (Software only)	**$99**	_____
		Total	_____

If you are paying by Visa/MC then fill out:

Card Name: ____ VISA ___ MasterCharge

Card Number: _____ Expiration: __/__

Name of Cardholder (Print) _____

Your Signature: _____

Phone # (in case of problems) _____

Notes:

Orders paid for by Visa/MasterCard will show up on your Visa/MC statement as a charge from Connectix Corp of Menlo Park, CA.

My records show that you purchased/updated your copy of The Debugger V2 & Nosy after September 1, 1989. You will recieve free updates in 1990.

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

The Debugger V2 and MacNosy - Release Notes - August 29, 1989

Updated on Oct 22, 1989

General

The major features of this release are:

Addition of the *Incremental Build System* for MPW users to speed up development of large programs (shipped with the Universal version only).

Improvements to the source level debugging for programs with MPW style ".SYM" files.

The Debugger now works with AppleShare/LocalTalk (twisted pair lines).

Numerous bug fixes that improve the usability and speed of The Debugger.

Special thanks to the many Beta testers who helped to make this release usable for the rest of us.

Note that the update sent to registered Mac II users is in "Auto UnStuffit" format.

Copy it to a **Folder** on your Hard disk and double-click on the Auto_Unstuffit Icon to unpack it.

Miscellaneous Fixes & Enhancements since Feb 28 th version

For a complete change history to The Debugger/Nosy since Jan 1, 1989 read the file "Mac II Notes".
Changes to The Debugger are:

- Add file "DRVRHdr.a" for DRVRs built with MPW so .SYM/map file is correct for Debugger.
- Handle C++ programs, "demangling" the names & handling names of the form "class::method".
  Current release demangles the constructor/destructor to "class::__ct" and "class::__dt".
  There are no special provisions for overloaded names.
- Add source level stepping for programs with .SYM files.
- Change single screen mode to Swap Screens versus Split Screen modes.
- In swap screen mode, allow either screen to be the main one (new version of xDbgr_Startup).
- Extend the type compiler to handle enumerated types, integer subranges, Pascal OBJECTs,etc.
  See the file "AppleTalk_EQUs" for examples.
  typName = OBJECT(className) fieldList Methods END; is the syntax for Objects.
- Allow the redefinition of types introduced in a .SYM file via **=TYPE,redef** in the ".dsi" file.
  Redefine your types to improve the display of them when a compiler "mangles" their definitions.
- Allow variable upper bound in last array of a record definition (Do a Cmd-space of TeRecord, etc).
- Fixed is now a base type that displays a longint as a flting pt number (16/integer_part,16/fraction)
- Expressions Parser - integers appearing in type declarations default to decimal base unless explicitly preceded by a '$". The "$" is NOT persistent as it is in arithmetic expressions.
- Record display - Add a kludge to auto recognize Color Grafports and new style TeRecords.
- Add "Dbg_Rsrcs" flag in .dsi files to turn off the recording/debugging of resources.
  This was done so as to minimize screen flashing when the user has only one screen.
- Add the "flag" BAD_ZERO to control the "garbaging" of Loc zero.
- Fix problems with not updating the current task in the menu bar display.
- Fix option-\ interrupt so that we end up in the task of the caller, and not Multifinder.
- Fix Go Until PC to undo the old TRAP 14 before setting a new one.
- Shutdown APPL command in MF mode - show name in menu bar and liberalize.
- Cmd-W of a local var name will bring up the "Local Vars-" window & highlight the name.
- Allow local and global variable Value windows to be dynamic (**Cmd-8**).
- **Note** - remember to turn Unloadseg Error Check **off** when debugging the MPW Shell or a Tool.
- Nosy - There are no globals in ROM.
- **Both - Work with MPW 3.0 and 3.1 SYM file formats, and run on a Mac IIci.**
- Cmd-shift-Minus of a decimal number will convert it to Hexadecimal
- Add AND, OR, NOT and DIV to the operators in the expression parser.
- Cmd-period to cancel a dialog or to abort reading of the .SYM file, etc.

How to Use MMU Memory Protection

MMU protection is selected or deselected at debugger startup time by the checkbox:

"Use MMU for Memory Protection, Screen Swap..."

If this option is checked, then The Debugger will grab extra space for the MMU tables, protect itself, etc.

The ROM.dsi file contains a new section, prefixed by "=U" that sets the default MMU Memory options for any task. It is similar to the "=F" flags section in the .dsi file.

If you wish to setup custom MMU options for a task, then copy the section from the ROM.dsi file to your .dsi file and modify it as per the directions in the section. You may check the results of your work by bringing up the "MMU Protection..." dialog in the **Stops** menu. The meanings of some of "=U" flags/options are:

When the **"Protect Debugged Tasks"** is box is checked, programs with a .SYM, .map file or Think C projects with run in protected mode, with the rest of the Mac protected as per the selections below it. This is the master switch for MMU Memory Protection.
The exclusion of programs with only a ".snt" file is intentional.

If the box **"Protect CODE Rsrcs in APP"** is checked, then the CODE resources (including the CODE 0 jump table) will be write protected.

The other options determine which areas of memory are write protected or read protected (implies write protection).

When an MMU Protection Violation occurs, you will get a message of the form:

```
"MMU Protection violation, attempt to Read(or Write Data at dddddd",
```

and a window titled `T_020_BCFSF` (020 Bus Control Fault Stack Frame) will appear.
Note that the value of the PC (Program Counter) has almost always be advanced at least one instruction past the instruction that caused the data fault
The address that you were trying to access is the `dddddd` in the "MMU ... message, and is also displayed as the value of the `dcf_addr` in the `T_020_BCFSF` window.

If you are looking at a source window, the use Cmd-click to change it into source and interspersed assembly code. It may be necessary to reestablish the PC by double-clicking on its value in the "-Registers-" window, and pressing Cmd-D.

Use the Ω-G (What block is an address value in) command to find out what block the data address or PC is in. The command takes an absolute address as its argument.

The Ignore Error (Cmd-5) command has been extended so that it will re-execute the Bus Error stack frame **without** MMU Protection. In this way you can let the write (or read) execute. If there is no outstanding Bus Error or discipline error, then Cmd-5 will do nothing.

Timing Differences

When a program to be debugged is running with MMU memory protection, the machine will be slowed down by about 25 - 30% due to the small page size. When no protected program is running, the slowdown is only about 8%.

Once more with feeling

In general, the appl may NOT access areas of memory outside this space, and this is what the MMU Protection attempts to implement. While writes to certain locations in the "system global" area (too numerous to mention) are tolerated by the protection algorithm, it is expected that executable resources inside the appl's heap do not write to the System Heap, etc.

As many custom MBDF's, WDEF's, etc violate this convention, you should use Rez or ResEdit to change their residence to the System Heap for debugging purposes. **MultiFinder is perfectly capable of expanding/contracting the System Heap as necessary, and the System will delete the resources when the program is finished.**

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

August 30, 1989

Dear Mac II Nosy/Debugger User,

Enclosed is the latest release of the Universal version of Nosy and The Debugger V2, and some accompanying notes. The major feature addition in this release is the Incremental Build System for MPW users. IBS received extensive beta testing this summer, which accounts for the delay in shipping this update. At this point in time I am fairly confident that most of the bugs have been worked out, and that many of you will find it useful.

The other direction that The Debugger has taken is to extend the limits of the size of the programs that it can handle by removing most of the limits of the sizes of the internal tables. Many of the beta testers have programs that exceed 500K. The Debugger breezed through them and one tester applied it to a new version of LabView which had 1.2 Meg of CODE resources! Some of you with large programs will notice that the second time I read in a .SYM file that it is instantaneous as I save a copy of the "digested" .SYM file away the first time I read it.

Another feature enhancement that you should find useful is the ability to redefine types defined in a .SYM file so as to improve their display format. Page 31 of The Debugger manual describes the basic data types and their display formats. To redefine types in a ".dsi" file (or window), start the section with "`-TYPE, REDEF`" instead of "`-TYPE`." Enter type definitions on the following lines, highlight the definitions, and Ω-F to enter them.

All of you should now have a copy of the revised Debugger manual ©1989 that is 69 pages long. If you don't, ask for a copy.

This is the next-to-last update for those of you who purchased The Debugger prior to July 1989. The last update in the Version 2 cycle will be mailed in November or December of this year. Timing will be a function of when I see header files for System 7 features.

Future directions that The Debugger/MacNosy will take include the following:

Additional features to support debugging of MacApp and Object Oriented Programs.
Turn Nosy into a source level code browser in conjunction with the .SYM file information.
Add "coverage" analysis (what procedures were executed) for use with SQA functions.
A Language sensitive (Pascal, C, Modula,..) editor for use on the Mac II.
Support for System 7 and virtual memory.

This update is shipped in "Auto UnStuffit" format. Create a folder on your hard disk to deposit the unstuffed files in. Copy the Auto Unstuffit file to that folder and double click on the Auto UnStuffit Icon to unstuff the files. Distribute the Debugger files, and install IBS as directed in the IBS release notes.

As a one-man band, I never have enough time for marketing. You can help me by "evangelizing" your friends and associates so that I can expand and hire programming help. The reverse side of this letter contains a Debugger comparison matrix. The Thumb comes from Think Pascal.

It you have problems with this release, give me a call. Feedback improves the breed.

Sincerely,

Steve Jasik

Steve Jasik **Compuserve ID: 76004,2067 Delphi: "MacNosy" AppleLink: D1037**

Features to be Removed in Future Versions

The Shutdown Debugger command (Files Menu) will be deleted in the future unless there is a loud outcry.

Known Problems

- Launching an APPL that resides on an Appleshare server no longer causes the workstation to hang, but now causes the server volume to unmount !!
- Do NOT try to use Discipline when Virtual is installed, the machine will hang.
- Transient resources such as Boomerang cause The Debugger's internal task table to get fouled up, if the Dbg_Rsrcs flag is on. Use Dbg_Rsrcs only as necessary until the problem is fixed.
- The Debugger does not handle .SYM file info for arrays with variable bounds.emitted by LS Fortran.
- Some programs do not like to run above 8 Meg, and cause strange blowups in The Debugger.
- The MMU Swap Screen option does not work on a Mac IIci if you have less than 8 Meg of RAM.

Future (Work in Progress)

We will be working on the following features for the next release (late May or June 1990):

- **Use the MMU to protect the machine from the Problem Program**, add hardware watchpoints, etc.
- Full compatibility with System 7, addition of trap calls and types, etc. (see below).
- Addition of a high speed performance analyzer, and "coverage" analyzer to tell you what parts of your program you did or didn't execute (useful for SQA types).
- Miscellaneous enhancements to trap discipline
- Enhance expression parser to allow writes to alternate windows, loops, etc.

MPW 3.1 Bug Alert for C programs

The MPW 3.1 Linker is about 3 times slower than the 3.0 Linker when producing .SYM files for programs that contain C compiled files. Apple has a fix for the problem. Contact MacDTS about the 3.2a10 Linker if you are experiencing increased Link times.

System 7 Notes & Hacker Alert

At present System 7 is in alpha, and it is a moving target. This version of Nosy and The Debugger work with System 7 after a fashion. There are some rough edges in them that will be cleaned up in the next release. Be aware that System 7 contains a data packing scheme for resources that causes problems for Nosy as it does not currently use the resource manager to read in resources. To disassemble a file that has packed resources, first use ResEdit to copy the CODE resources to another file, and then paste them back into the original file. This removes the packing of the resources.

Have Disk, Will Travel (available for weddings, funerals, bar-mitzvas, etc)

I am available for personal appearances before user groups of a suitable size, companies interested in a site license, etc.

Shows I will Attend

I will be at the SF Macworld in April (leave messages for me at the MacTutor booth) and the Apple Developers conference in May. I will give out updates on an exchange basis, so bring your labeled Debugger disk with you.

DA's worth having

The Inside Mac DA by Bernard Gallet is quite useful. To obtain a copy, send $20 to him at 600 MIller Ave, Cupertino, CA 95014.

For Sale - Fast Text Search Technology

An algorithm that is linear in the number of characters on the first pass, and builds a database equal to 8 bytes per "line" of text. On succeeding passes over the text the algorithm time is linear in the number of lines. Call if interested.

If you have problems with this release, give me a call. Feedback improves the breed.

Sincerely,

Steve Jasik

Steve Jasik **Compuserve ID: 76004,2067 Delphi: "MacNosy" AppleLink: D1037**

The Debugger V2 & MacNosy - IBS Version 1 Notes - 8/28/1989

General

The release version of the IBS System incorporates fixes to bugs found by the beta testers. To use the IBS System you must have a Mac with at least 4 Meg of RAM. The Debugger must be running in the background with MultiFinder as the startup. The MPW Shell must be running.

The goal of the *Incremental Build System* (IBS) is to reduce development time for MPW users by reducing Link and compile times. The main component of IBS is an Incremental Linker that works in conjunction with *The Debugger*. It will let you Link in changed modules (procedures or functions) into a running program. With the MPW Shell running, the sequence of steps is that you start up your program. While you are debugging your program, you can "park" it in a "safe" place (after a _GetNextEvent/_WaitNextEvent call) in *The Debugger* and transfer to the MPW Shell via the "Side Door to MPW" command in the "Apple" menu. Once in the Shell, you may modify your program to change existing procedures or add new ones, and rebuild them with the "PatchBuild" script. The final step of The PatchBuild script will do a *PatchLink*, and transfer you back to *The Debugger*. It will install the patches and adjust the ".SYM" info so that you can debug your changed modules. This process can be repeated ad infinitum as long as you **do not change the interface sections** of your files. More on this later.

The purpose of IBS projects is to maintain the list of modules that have changed since the last "master" or full Link. IBS projects have no relation to Projector projects.

IBS commands

The file "**UserStartup•IBS**" adds the new commands that you will need to use IBS.

To **create** a project, use the "`Create Project`" command to select the copy of the APPL or Tool that you are developing in the folder that contains the ".SYM" and ".map" files.
Note that PatchLink (the incremental Linker) needs a "-L" style map file to do its magic.

Because we do not have a language sensitive editor, **you must tell IBS what modules (procedure/function) you have changed** since the last "master" Link that you did with *make*. **You do this by hilighting the name of the changed or new procedure/function in the window containing the source file** and issue a "`Add Hint`" command (Ω-Z).

To **view** the list of changed modules in a IBS_Project, use the "`Show Proj Status`" command. It will write the list of changed modules and implementation dependencies to the front window.

To **transfer back** to *The Debugger* you may use the" `Resume PP`" command. You may also use the option-\ to transfer from MPW to The Debugger, but you will not be able to activate the program you Suspended until you do a "`Resume PP`".

In order to **reset** the IBS project information during a **full make** of your APPL/Tool, insert a "`Execute IBS_Reset`" line into your build script **prior** to the Link command, or use the Reset Project command from the IBS menu. The IBS_Install script does this for MacApp V2.0ß9 users.

The shell variable "**IBS_Project**" holds the name of the current IBS project. When you start MPW it will be set to the first project (second line) in the file "IBS_ProjInfo" (System Folder). The list of known IBS projects will appear at the bottom of the IBS menu. You may use these menu items to switch between projects if you have more than one IBS project.

Use the" **`PatchBuild`**" command (Ω-B) to compile and PatchLink your changed modules or create a special *make* file to perform the task PatchLink will issue a Resume PP command as its last action if it is successful. The PatchBuild command creates a "PatchBuild.doit" script which you may reuse as long as you confine your changes to the current set of modified files that PatchBuild has created compile commands for.
The Patchbuild script will Save all your open files (windows) as part of "PatchBuild.doit".

The "**To Dir of Top Wind**" command (Ω-D) changes the current directory to that of the top window. It is similar to its counterpart (Use Task) in *The Debugger*.

C programs - Some considerations

The compile command for C should contain "-b" so that string literalss are allocated in the segment they are referenced in and do not occupy (valuable) space in the global area.

More Details

The IBS system handles modules whose entry points are at the beginning of the module (normal for the C and Pascal compilers). For those of you who wish to change procedures written in assembler, they must conform to the rule that each procedure be preceded by a PROC or FUNC directive, and that it is the only entry point in the module.

The SIJ_Proj tool is used to implement most of the commands in the IBS menu. Pascal programs where the sources for the implementation is in different folders than the interface files have to add some "**Implementation dependencies**". To do this, issue commands of the form:

```
SIJ_Proj -i "{IBS_Project}" "Implementation_directory" "Interface_directory"
```

IBS adds a default Implementation dependency rule to all projects for MacApp 2.0ß9.

File nameing conventions pertaining to the interface/implementation file suffixs are discussed in the help file (:Scripts:IBS_CP_Help) associated with the Create project command.

As stated above, when you change the Interface section of any file in your "project," then you must use *make* to rebuild your APPL/Tool. In practice the actual conditions are somewhat less restrictive. In Pascal programs you may add global variables to a unit only if you have left a slop area which you can subtract from. You may add types if you are willing to feed *The Debugger* the type information via ".dsi" files. You can change the interface to a procedure if you change all the calls, and mark the calling routines as changed via Add Hint commands.

Beware: The IBS System only knows about the files and procedures that you tell it about. If you change the interfaces and don't do a Make, then your time may be wasted.

Environments

When I first saw MPW version 1 I was somewhat appalled by the fact that the Apple programmers had taken a 20 year step backward and picked the Unix Shell as the model environment for us to program in. After all Lightspeed Pascal accompanied the introduction of the Macintosh, and it contained a language sensitive editor and an integrated make tool that did not require one to specify the obvious. Integrated environments and language sensitive editors have been discussed in the computer journals (SIGPLAN proceeding, etc) for over 10 years. The problem of creating and maintaining a large program requires that one treat the source as a database and the objects in it are the names. One advantage of such an approach is that queries to find all references to a name or its definition are fast. Another is that incremental compilation and linking can easily built into such a system and made invisible to the user.
At this point I am negotiating to acquire a language sensitive editor for C, Pascal and Modula that will work in conjunction with MPW. In addition to getting rid of the Add Hints command it should also make it possible to implement a database approach to program development.

PatchBuild versus Make

One of my friends who races cars and other objects containing gasoline engines commented on the difference between automatic and manual transmissions by noting that the former were like artificial insemination. There are some things that a man should do himself!
I use the MPW *make* program as little as possible. I keep a window open that has compile commands for all the files in *The Debugger*. The Link and Rez commands are surrounded by parens so I can click on the outside of the opening paren and select the text to be executed.

In the case that PatchBuild does not measure up to the task at hand, you may modify the PatchBuild script so as to spit out the appropriate compile commands by adding sequences of the following form (example assumes the C compiler).

```
IF "{leaf}" == file_a
  ECHO "C file_a  Options_for_file_a  -o ∂"∂{objFldr∂}:∂"" >>"{out_File}"
ELSE IF  "{leaf}" == file_b
```

"Source Level Debugging" - Think's LightSpeedC Projects

For LightSpeedC projects compiled with **"Use Debugger" on**, *The Debugger* will display the source code of the procedures in windows. One may optionally toggle between source and interspersed ASM with a Command-Click in the content region of the window. The types of the global variables may be supplied by the user in a ".dsi" file. The names of the local variables may be inferred from the ASM display.

Source Level Debugging - MPW V3 Pascal & C, TML Pascal II, LS Fortran, etc

The source level debug information is relatively complete. See page 35 of the manual for notes on source level debugging. The generated code from the new MPW C compiler looks good.

MPW Goodies & IBS

The **MPW Goodies** folder contains two tools that can be used to implement a inter file GOTO marker scheme ("ctags"). The code was rewritten for this release to work in conjunction with IBS. Future releases of IBS will use the master marker file to help PatchLink resolve unsatisfied external references. For now, you can use them to go from the reference of a procedure/function to its definition.

Future Plans (or it didn't make it into this release)

Items scheduled for future releases of The Debugger & MacNosy include:

Support for LS Fortran common variable and array info in the ".SYM" file.
Parser extensions to handle local variables, looping constructs (FOR,...), Trap calls, etc.
Extend trap intercept facility to be able to specify trapnames & selectors.
Ability to display globals in Rsrc's that are based off A4.
Add "Pert" type displays of the program call graph (tree of procedures).
Extend IBS to resolve "unsatisfied external references" .
Make *The Debugger* operate in a 32-bit environment and work with Virtual memory schemes.

Known problems: MPW 3.0 (fixed in 3.1 ??)

MPW Pascal does not put out any type information for objects (MacApp).
MPW C does not put out proper type information for Enums.
MPW C creates multiple associations for variables that it assigns to registers in a block without specifying the range of the association in the block so that one has to inspect "-Local Vars-" and the object code to decide which assignment is valid.

Known problems: The Debugger

The Debugger is sensitive to the presence of certain INITs and will not start up in background mode if they are present, **or load before *The Debugger***. This is a continuing problem. Known troublemakers are flagged in the "-Notes-" window. You can use a "binary" search to figure out which INIT is causing *The Debugger* a problem, and send me a copy of it so I can fix the problem.

The Debugger does not start at boot time if a Radius Accelerator is installed. This is a known problem, and has defied most of our efforts to fix it. Read the file "Radius Notes".

I have seen one case where *The Debugger* did not boot when the standard 6.0.3 MultiFinder was used. Use the "set aside" MultiFinder (6.1b7) instead.

When The Debugger is loaded at boot time and one or more INITs are loaded after it (zSuitcase), you will get the "ROM patches adjusted" message in The Debugger no matter what you do. **Ignore it !!**

Oops, The following didn't make it into the Manual:
To pass parameters to a "?UserP", use assignments to global variables in the "action clause" that calls it.

The Scroll bar in the "**Type@nnn**" windows is a "page" type scroll bar that does a page up/down when clicked in the gray region. It is useful in the case you have an array or record. An example is to define the type: `trapTbl = record a,b,c,d:Array[0..3] of ProcPtr end;` and do a `"trapTbl@400"`.

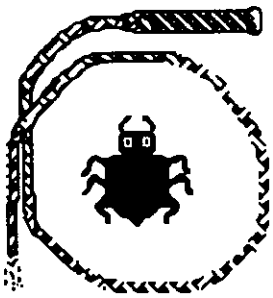
Your feedback is always appreciated. Many of the features in *The Debugger* were suggested by users, IBS, the ability to redefine types introduced in SYM files, etc are examples.

Information

Debugger Tech Note #0

Installing The Debugger

November 89



Control

About Debugger Tech Notes

Debugger Tech notes supplement the Debugger Manual (little gray book).
Jasik Designs is basically a one-person business, and I wear most of the hats. After answering your tech support questions for a number of years it has become apparent to me that most Macintosh users with a new software package in their hands are like a dog digging for a bone. They rip off the shrink wrap, and everything but the floppy disk goes over their shoulders. The Debugger has its quirks, and attempts to use it without becoming aware of some of them can be frustrating. **RTFMQR** - Read the Fine Manual and Quick Reference for maximum pleasure.

This tech note is the latest in an attempt to make the process of installing The Debugger as painless as possible. It is a rewrite of page 7 in The Debugger manual. Pages 8 and 9 discuss some of the finer points in more detail.

1. Make a backup of ***The Debugger*** distribution disk (use CopyTo Mac or its equivalent).
2. Copy the contents of the distribution disk (the White one) to your hard disk.
   If the disk does not copy properly, then check it for media errors, and if it is bad, then return it to me.
   The Debugger must reside on the boot (startup) volume .
3. Copy the files in the **put files in System fldr** folder to your System folder. If you will not be using The Debugger in **background mode**, then remove **xDbgr_Startup** from the System folder. The file called Macsbug in this folder patches _SetTrapAddress. It is necessary for the operation of The Debugger. You may rename it "disassembler" if you wish to run Apple's Macsbug in the background, but be aware that Apple's Macsbug will ONLY debug The Debugger, so you will be wasting about 100K.
4. Create a folder in your System folder and copy ALL your third party (non-Apple) INITs into it. This is to remove any potential INIT conflicts from the initial installation process. Once you have become familiar with the operation of The Debugger, you can experiment with moving the INITs back into the System folder. Some of them will have to be prefixed with a "z" so that they load after The Debugger and do not interfere with its operation. The basic rule is that if the functions performed by the INIT can be used in The Debugger, then it is a candidate to be loaded before it (i.e., AppleShare).
   The following INITs are incompatible with The Debugger (may not be in your System folder):
   InitPicker, MacroMaker, Easy Access, TMON, Flow Fazer (and some other screen savers)
   The following INITs must **load after** The Debugger (prefix them with a "z"):
   Tops Version 2 (and Softtalk), SuperClock, II Display (Radius), InBox, SuperLaserSpool, SuitCase
   I have had problems with HierDa (aka DAmeuz) by jbx on occasion, and removed it.
   One user has reported that to work on a IIci SuitCase II ß6 had to be loaded before The Debugger.
   You should keep in mind that the above lists of problem INITs are incomplete.

4a. Mac II users with 2 or more monitors: Select the screen that The Debugger will occupy.
   Bring up the monitors display in the Control Panel. Hold down the option key and you will see a smiling Mac icon in one of the monitors. That monitor is the startup screen, which The Debugger will occupy. You can move the smiling Mac to the monitor of your choice.
   Do NOT attempt to run The Debugger on a monitor that is driven by a 24-bit video card. Get a second monitor with a 1-bit video card.

Debugger Comparison Matrix

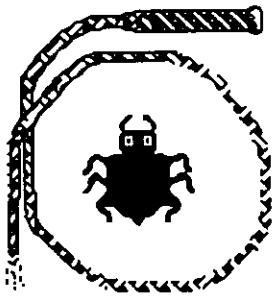
Feature	The Dbgr	SADE	TMON 2.8	MacsBug 6	Think C	Think Pascal
Macintosh Interface	Yes	Yes	👎	👎	Yes	Yes
TextEdit operations, Cut, Copy Paste	Yes	Yes	👎	👎	Yes	Yes
View/Edit Text files	Yes	Yes	👎	👎	Yes	Yes
Multiple windows	Yes	Yes	limited	👎	Yes	Yes
Source Level Debugging	Yes	Yes	👎	👎	Yes	Yes
View Corresponding ASM	Yes	👎	Not applic	Not applic	👎	👎
Assembly Level Debugging	Yes	Yes	Yes	Yes	👎	👎
Symbols from .map files	Yes	👎	Yes	👎	👎	👎
Symbols from Think C Projects	Yes	👎	👎	👎	Yes	Not applic
Symbolic Debugging of Multiple Tasks	Yes	Yes	limited	limited	👎	👎
Breakpoints	Yes	Yes	Yes	Yes	Yes	Yes
Breakpoints with conditions	Yes	Yes	👎	limited	limited	👎
Trap Intercept	Yes	Yes	Yes	Yes	👎	Yes
Display Args to Trap in natural Format	Yes	👎	👎	👎	👎	👎
Displays Patches to ROM	Yes	👎	👎	👎	👎	👎
Memory Watchpoint (TracePoint)	Yes	👎	👎	limited	👎	limited
Discipline & other Error Checking	Yes	👎	No Color QD	👎	👎	👎
Structured Display of Data	Yes	Yes	limited	Yes	Yes	Yes
Debug Time Definition of Structures	Yes	👎	👎	👎	👎	👎
Allows Typecasting of Data	Yes	Yes	limited	Yes	Yes	Yes
Continuous Animated Step Mode	Yes	Yes	👎	👎	Yes	Yes
Emulation Trace	Yes	👎	👎	👎	👎	👎
Has Stack Crawl Display/Window	Yes	Yes	👎	Yes	Yes	Yes
Has Annotated Heap Display	Yes	Yes	Yes	Yes	👎	limited
Has find Heap Block that address is in	Yes	👎	👎	👎	👎	👎
Installs at INIT time	Yes	👎	Yes	Yes	👎	👎
Disassembles 680x0 instructions	Yes	Yes	👎	Yes	Not applic	Not applic
Reads "Script" files	Yes	Yes	👎	👎	👎	👎
Expression evaluation	Yes	Yes	Yes	Yes	Yes	Yes
Courtesy Bkpt at first Instruction	Yes	👎	👎	👎	Yes	Yes
Symbolic Debugging of Code Resources	Yes	Some	👎	👎	Some	👎
Modify Data in memory	Yes	Yes	Yes	Yes	Yes	Yes
Symbolic Assembler	👎	👎	Yes	👎	👎	👎
Has Go Until PC	Yes	Yes	👎	Yes	Yes	👎
Has Set PC	Yes	Yes	Yes	Yes	Yes	👎
Has MacNosy	Yes	👎	👎	👎	👎	👎

Information

Debugger Tech Note #1

The Redefinition of Types

Sept 89



Control

This Tech Note is primarly for programmers who use MPW 3.x.
The Type information in ".SYM" files gets mangled by the compilers during compilation. Part of the problem lies in the compilers' inability to let us programmers say what we mean, and part of the problem lies in the fact that the compilers' lose the simple "name" equivalences that we make.

The features of The Debugger which are of interest to us are:

1) The Debugger contains a rich set of "base" types such as word, HLongint, RefNum, etc that allow one to control the display format of the fields in a record (or C struct).

The Pascal type definitions ("name" equivalences) for these types are:

```
HLongint    = Longint; { display a 32 bit integer in Hex format }
DateInSec   = Longint; { display a 32 bit integer in MM/DD/YY hh.mm.ss format }
Fixed       = Longint; { display a 32 bit integer in Floating Pt format }
word        = integer; { display a 16 bit integer in Hex format }
RefNum      = integer; { display a 16 bit integer as a FCB RefNum, (Hex & file name) }
```

2) The Debugger lets one define a record with a final Array that has a variable upper bound. For example:

```
TERec = Record  { 100 bytes}
{   0} destRect         : Rect;
  ...
{  90} caretHook        : ProcPtr;
{  94} nLines           : INTEGER;
{  96} lineStarts       : Array[0:nLines] of INTEGER;
END;
```

3) One may define objects in The Debugger by using the Object Pascal Object definition:
objName = Object fieldList methods END;
or objName = Object(className) fieldList methods END;
where className is the name of a previously defined object class.

One may use these facts and the ability to **redefine** the types defined by the ".SYM" file to enhance the format of information displayed in the data value windows ("type@nnn").

For example, take the case of Bernard, whose types were getting mangled by the C compiler. One record definition came through as:

```
VarElem = Record  { 202 bytes}
{   0} stPara           : Longint;
{   4} endPara          : Longint;
{   8} isHotLink        : SignedByte;
{   9} name             : Array[0..128] of SignedByte;
{ 138} format           : Array[0..63] of SignedByte;
```

does not have strings built into it, and in this case The Debugger did not pick up the fact that `name` and `format` are really strings. While Bernard can program in Pascal, this program was written in C, so the Pascal definitions were not available in the source code. But they are available in Pascal format in The Debugger.

Installation Procedure (for MPW 3.0 release &MPW 3.1ß1 dated June 19, 1989)

Copy the "**IBS V1 for MPW 3.x**" folder (on the Blue disk) to your hard disk. Open it up and double click on the file **IBS_Install**. Read it and execute it (Cmd-A and Enter).
Quit from the Shell, and restart it from the Finder.
Open up the **IBS_Verify** file and execute it. If the Shell patches are working, then it should terminate without an Alert dialog. Use **ResEdit** to change the MDEF's in the MPW Shell.
You may wish to add the file "**add to MPW.Help**" to the end of the MPW.Help file.

The file **U_InHints.p** is for compiler writers who want to implement conditional compilation.
The "**Shell Patch Source**" folder contains the necessary files & Tool to generate a version of the MPW Shell .DPF file for later versions (3.1) of the MPW Shell.
The penalty for renaming the MPW Shell is that weird things may happen.

New Stuff - Some Dirty Details

DPF files (Debugger Patch Files) were created to dynamically patch an APPLication or Tool when The Debugger is present (running in the background). The Shell .DPF patch file allows us to copy the original version of a file that is in a window and may have been modified. This is necessary as the source/object code information in the ".SYM" file was built from original.
The Add Hints command will make a copy of the source file with a "_old" extension if necessary.
The Reset command will delete the "_old" files.

IBS does not keep a complete list of the source files in your project. It is imperative that you do an Add Hint when you change a file before trying to save it. The penalty for failure is that the source displays in The Debugger will get out of synch.
Once you have done an **Add Hint for a file** you may pound on Cmd-S all you want.

The Pascal compiler has been patched to do incremental compilation via a .DPF file. As we only want this option to be active when we are doing partial compiles, I choose to make a separate copy of the compiler named "**XPascal3**" to which the patches apply.

The IBS creates numerous new files & folders with the following extensions:

.bmi	Binary Map info, created by PatchLink, a binary version of the ".map" file.
.PLI	Patch Link Info - Partially Linked modules. Created by PatchLink.
_PLI.SYM	The partial SYM file that PatchLink passes to *The Debugger*.
_IBS.proj	The file that holds the IBS project information (list of changed files & modules).
ProjInfo:	A folder that holds the object modules from the incremental compilations.
IBS_ProjInfo	A text file in the System folder that holds the list of IBS projects.

Getting Acquainted - Once around the block with TeSample

Having installed the Incremental Build System, and run the IBS_Verify, let us try to use IBS.
From MPW go to the PExamples folder and find the make file for TeSample. Add "`-SYM ON`" option to the Pascal command. Add "`-SYM ON -L >TeSample.map`" to the Link command.
Build TeSample (this is the "master" Link), and Launch it (from the Finder or MPW).
Once it has started up, enter Option-\ to transfer to *The Debugger*. It should bring up a display of TeSample's mainEventLoop.
Use the "Side Door to MPW" command to transfer to MPW.
Select "Create Project" from the IBS menu and select TeSample from the PExamples folder as the IBS project. Open up "TeSample.p" and find the Function `isDAWindow`.
Add the line : `DebugStr('Hello from PatchLink/IBS');` after the `BEGIN` line.
Hilite the procedure name and select the "Add Hint" command from the IBS menu.
Select the PatchBuild command from the IBS menu.
If everything goes as it should the XPascal3 compile and the PatchLink should complete successfully, and PatchLink should transfer you back to *The Debugger*.
From *The Debugger* do a Cmd-E (exit to Program). You should almost immediately return to *The Debugger* and be looking at the new version of `isDAWindow`.
The message `'Hello from PatchLink/IBS'` should be in the -Notes- window.
Note: Your patched code is in a a non-relocatable block 32K long that is after the Preloaded resources (CODE 1, etc). It will be labeled "Debugger PLI buffer" in a heap display.

Change History from 9/89 to 3/90

summary of external changes to The Debugger and Nosy between September 1,1989 and March 1,1990.

General Changes to Both Nosy & The Debugger

- Fix to work with IIci & Portable
- Revise Type processing so we can handle upto 8192 user types in a .SYM file
- Fix bugs associated with processing files on multiple volumes
- Change Cmd-G so that search is case insensitive, if Cmd-shift-G then case sensitive/token search
- Add - select Line when triple Click is detected
- Do balanced match hiliting for ({ [' " ` and / (a la MPW Shell)
- set preload bit for some commonly used dialogs (find,Cmd-D dialog)
- Fix - Change name of the low memory loc BasicGlob to ExpandMem & add record type for it
- Change names of all sij/ aux files to "sij_"

Changes to The Debugger

- Finder and MultiFinder .snt files **are now optional**
- Swap Screen mode now works with screens that have more than 1 megabyte of screen memory
- Step continuous in Swap Screen mode uses the MMU to remap the screen so there is NO screen flashing, **TRY IT.** Remember: **Cmd-shift-Semicolon** to bypass the Step Continuous dialog.
- The Debugger now works with the Virtual and Maxima INITs (from Connectix)
- Allow $Annn syntax in Cmd-J to set trap intercepts
- Hiliting a Dbgr flag name and Ω-F Toggles the value of the flag
- Add flag Dbg_ThC to control the auto Debugging of Think C projects
- Add display of Control & MMU registers for 680x0
- Fix - bug in Read .snt file proc that caused many crashes and other unexplained symptoms
- Fix - Debugger can now handle programs with up to 254 CODE segments
  Fix - always patch __setjmp so that one can Step thru it (MPW only)
- write(var) where var is a string, array or record displays the object in its proper format
- Fix - allow both string; and string[n]; in type declarations
- Add - ?Rec(@GlobalVar,addr) where GlobalVar:String; holds the name of the type
- Fix - change MacApp class display to show class hierarchy
- Fix - WatchPoint - allow fba to be any address

Nosy changes

- Add "s1" option to Seg_Ref map in tty Maps menu to list 1 ref interseg calls (**TRY IT**)
- demangle C++ Macsbug names
- if file name contains the string "Macsbug" then disassemble the Data fork
- fix - recognize Macsbug names in ROM patch area

IBS Changes

- Add an Interfile goto marker facility, and scripts to set it up (see Tech Note #4)
- fixes to PatchLink to process MacApp names properly,
  correctly process multiple changes in multiple files, etc
- Add Object list collector to MacApp, Object Inspector Menu to Dbgr, etc (see Tech Note #2)
- fix - UserStartup to disallow Hinting on names that contain spaces, etc
- fix Delete Proj script to use GetListItem of IBS_ProjInfo file
- Add patch files for MPW 3.1 final, and make changes to IBS_Install script.
- Change MABuild mods to auto create the IBS project, and touch the .SYM file

xDbgr_Startup, RunDbgr

- Revise internal format of info passed to The Debugger
  Ignore Inactive bit of second Screen, and plunge ahead anyway
- Remove Use Time Mgr msg & replace with USE MMU to control Screen Swapping

OR by creating a special make file with dependency rules for you compiles as follows:

```
APPL.PLI  ƒ  source_file_list_A
    compile_command_A

APPL.PLI  ƒ  source_file_list_B
    compile_command_B
    ...

APPL.PLI  ƒ  "{IBS_project}_IBS.proj"
    PatchLink  ∂"{IBS_project}∂" ∂"∂{objFldr∂}:∂" -wf -l
```

Q & A time with the Head Nose

0) The Debugger won't show me my source.
Yes it will, read page 34 of the grey manual, and make a ".dsi" file with a "=Source" section.

1) My Source is not in Synch when in the Debugger.
I'z told you to do an Add Hint before you pounded on the Save command and you forgot.
Do not pass GO, go back to MPW and do a full Link, OR fixup xxx_old so you are in synch.

2) Not all the procs I modified were linked in.
Are you sure that you did an Add Hint ?
Check the mini-Link map in the Patchbuild.doit window and do a "Show Proj Status" (Ω-P).

3) I have a BIG C program and get the word from *The Debugger* that I have "Too many Types"
See, I told you that Pascal was a better language and you just didn't listen.
Seriously, this is a known problem with the 3.0 C compiler. It should be fixed in 3.1 release version.
For the moment you will have to compile some units with -SYM ON,NoTypes or such,
change anonmyous Struct definitions to named ones, and centralize your type definitions in header files.
An anonmyous struct definition is one of the form : Struct { type name; ... } name .

4) What is the earliest time after I launch a program that I can do a Side Door to MPW ?
I haven't figured that out yet, but it is sometime after the first _GNE/_WNE.

5) I am building a MPW Tool, what special actions should I take ?
If you don't use PatchBuild, then add '-nr' to the PatchLink line in your make. Your patches will be added to the Tool the next time you run it. Don't even think of trying to use the Side Door command, The Debugger won't let you. Attempting to have the Shell task both active and suspended at the same time is equivalent to trying to disappear up your fundamental.

6) Why do I have to do a full Link if I add/subtract globals, objects, types, etc ??
Because Rome wasn't built in a day, and neither was this system.

7) I'm using a preprocessor that creates ".b" files that are translated into ".c" files that are compiled. Help !!
Play with the PatchBuild script. Change the lines :

```
ELSE if "{PFile}" =~ /≈.c/
  echo "C   ∂"{PFile}∂"  {IBS_Coptions} -o ∂"∂{objFldr∂}:∂""
```

to

```
ELSE if "{PFile}" =~ /≈.b/
  echo "Compile_B   ∂"{PFile}∂"  {IBS_Coptions} -o ∂"∂{objFldr∂}:∂""
```

Where `Compile_B` is the script that preprocesses and compiles your ".b" files to ".b.o" files
so that the "SIJ_Proj -B" command makes the correct test.

8) I have a big MacApp program and I can't get it to Link or The Debugger complains about not enough space to read the .SYM file or some such garbage, Help !
If you don't have 8 Meg, then reach into your pocketbook and invest in another 4 Meg. RAM prices are down, down down, so splurge ! Alternate solutions are to reduce the size of RAM cache 64 - 128K and the MPW partition to 1224K. Use the "-mf" option on the Link directive & "flush" the tool cache before Linking (done by IBS_ResetProj).

9) Why does the MA_IBS_options have a NoAlign in it ?
I removed it at the advice of the MacApp group which pointed out that some of you would be using C++ in the near future. You may add it back if your program is going to stay in Pascal.
It was put in so your global variables are in one block that you can play subtract/add games with within a unit, and can avoid doing a full Link.

10) What about partial compilation for C?
Possible in the next release.

11) Do IBS work with C++ ?
Yes, hilight the class and method name ("class::method").

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

The Debugger V2 and MacNosy - Release Notes - Feb 28, 1989

General

The major features of this release are source level debugging for programs with MPW style ".SYM" files, partial source level debugging for Think C project files, a revised and expanded manual that includes a tutorial section, and a quick reference sheet. My apologies for the delay in getting this release out, but circumstances beyond my control (the MPW Linker) delayed it.
I will be at the Apple Spring Developers conference, remember to bring your labeled disks with you.

Miscellaneous Fixes since Nov 7 th version

Most fixes are to The Debugger.

- Make background Operation of Debugger "silent" if Doc_Msgs = 0;
- Reverse swap all of LVL2dt so debugging of Serial Drvr & AppleTalk works.
- Debugging of "cdev's now works "fileName/cdev_F020.map".
- Most commands - allow references to names in code (ASM or source) windows.
- Patch Get1NamedResource and GetIndResource so XCMD's from files and 4DEX procedures work.
- Change **Cmd-comma** for RAM calls to Go Until so MacApp Works.
- Fix Discipline for some Color Traps to Eliminate spurious Errors.
- Cmd-W of a local var name will bring up the "Local Vars-" window & highlight the name.
- Allow local and global variable Value windows to be dynamic (**Cmd-8**).
- **Note** - remember to turn Unloadseg Error Check off when debugging the MPW Shell or a tool.
- Nosy - fix problems with the ROM explore when patches refs CODE above BufPtr.
- Fix problems with Dup Macsbug names in explore (Illustrator 1.1).
  Both - ".snt" names, suffix the Rsrc_id as a Hex number instead of a Decimal ID.
- Cmd-period now cancels dialogs also.

"Source Level Debugging" - LightSpeedC

For LightSpeedC projects compiled with "Use Debugger" on, *The Debugger* will display the source code of the procedures in windows. One may optionally toggle between source and interspersed ASM with a Command-Click in the content region of the window. The types of the global variables may be supplied by the user in a ".dsi" file. The names of the local variables may be inferred from the ASM display.

Source Level Debugging - MPW V3 Pascal & C, TML Pascal II, LS Fortran, etc

The source level debug information is relatively complete. See page 35 of the manual for notes on source level debugging. I have successfully tested the source level debugging on a 300K application compiled in MPW C that had a 665K ".SYM" file, and on a large application written in Pascal. The generated code from the new MPW C compiler looks good. The bad news for users with large MPW C compiled applications is that the C compiler puts out a large amount of unnecessary TYPE information for the ".SYM" file that the linker does not squeeze out. The number of types appears to be a function of the number of UNITs (files). This causes a problem for me because I only allow 2047 user types. A band-aid fix is to compile with the parameter: "-sym ON,noTypes" for some files to reduce the amount of type information. *The Debugger* displays the number of types in the ".SYM" file on the line after the " reading xxx.SYM" line. The MPW group has been working on fixing this problem in the Linker, and a revised version of the Linker should be available via AppleLink or such.

You can help to expedite a fix for this problem by voicing your dissatisfaction to Apple.

While it is not a problem, you should note that I give Anonymous Records names of the form "xx?".

5. Run Nosy and create **.snt** files for the Finder and Multifinder. **This step is optional for versions > 11/22/89**

   Now most of you don't want to read the Nosy manual, so here is a quick how-to-run Nosy lesson.
   Double click on Launch Nosy. A SFGetFile dialog box will appear. Navigate your way to the proper folde and select a file to disassemble. Answer the query "list resources [n] ?" with a return, or y and return if you wish to view the resources in the file. To disassemble the CODE resources in a file, reply to the query "disassemble resource type [CODE]?" with a return. When the TreeWalk options dialog box appears, press return. Nosy will do its thing, and after a few seconds or minutes in the case of ROM or a very big program you will end up in window mode. Select the menu command "save .snt, reRead .aci" from the Misc menu. Reply to the dialog with a return.
   Now exit from window mode with a Cmd-Q and you will be looking at a dialog box with 4 big buttons. Select one of them.

6. Create OR customize **ROM/CODE.snt** in the **Dbgr/Nosy Files** folder to your system configuration.

   For the **Universal** version of ***The Debugger*** the default **ROM/CODE.snt** is configured for the 256K ROM Mac II family machines (II, IIx, IIcx and SE/30). You may customize it if you wish. To set up on any other machine (IIci, +, SE or portable, etc), you should answer "**n**" to the bypass treewalk query in Nosy and follow the procedure in Step 5.

   If you have the Mac Plus/SE version of ***The Debugger***, the **ROM/CODE.snt** file is configured for the Plus. The **ROM/CODE.snt** for the SE is in the folder, "**Mac SE files**". If you are installing on a Mac SE, drag the **ROM/CODE.snt** from "Mac SE files" to "Dbgr/Nosy Files".

   To adjust the ROM/CODE.snt file, boot your machine (to install the special MacsBug), and bypass ***The Debugger*** by pressing the "**Cancel**" button in its opening dialog. Run Nosy to disassemble the file called **ROM** in the **Dbgr/Nosy Files** folder and produce your own **ROM/CODE.snt.**

7. Boot the Mac again, and this time press the "**Launch The Debugger**" button in the opening dialog. Problem areas (INITs, insufficient memory, etc) are discussed in pages 8 and 9 of the Debugger manual.

8. **MPW users**: Copy the file InformDbgr to the "{MPW}"Tools: folder. If you do not have, or are not using the Incremental Build System, then add a line to your UserStartup which executes InformDbgr.
   If you will be debugging MPW Tools, then you must disassemble the MPW Shell and create an ".snt" for it.

9. Make a backup of your System file, Finder and MultiFinder.

Debugging is dangerous business. When you do a Flush Volumes and Boot in The Debugger or the Finder, the system will close all your open files. I have encountered situations where the system has silently trashed MultiFinder in such a way that it appears to be operational, but it caused malfunction of The Debugger. By planning ahead, you can save yourself alot of grief.
Here is the procedure:
Make a folder on your hard disk, and option-copy copies of the System file, Finder and MultiFinder into that folder. If The Debugger or your system starts acting up, then you can move the current Finder, MF and System files to the desktop (gray area), and copy the saved files into the System folder and boot.
If the problem goes away, then trash the bad files, and continue.
See also "Hard Disk Hints" on page 9 of the gray manual.

10. Install the Incremental Build System (per the instructions in the IBS Notes), if you are using MPW to develop Applications.

How I Debug The Debugger (for the curious).

The Debugger is just another Application, that is loaded at INIT time by xDbgr_Startup. To have one copy of The Debugger debug another, one can do the following:
In the Finder, set Finder as the startup, and boot the mac with the first copy of The Debugger launching into the background at INIT time. Make a second copy of The Debugger and change its name to "The_Debugger". Disassemble it in Nosy to create an .snt file.
Use RunDbgr to launch The_Debugger and in RunDbgr, set the startup APPL to be MultiFinder.
I can track down about 98% of the bugs in this manner, for others that are address sensitive, I have to use one of the other debuggers.

Future plans (or it didn't make it into this release)

Items scheduled for future releases of The Debugger & MacNosy include:

Name "demangling" for C++, and whatever else is needed for it.
Support for packed fields, LS Fortran common variable and array info in the ".SYM" file.
Parser extensions to handle local variables, looping constructs (FOR,...), Trap calls, etc.
Extend Type definition language to handle C syntax of type definitions.
Extend trap intercept facility to be able to specify trapnames & selectors.
Implement stepping at the source level.
Extensions to handle variable length records in the type language.
Ability to display globals in Rsrc's that are based off A4.
Make *The Debugger* operate in a 32-bit environment.
To be nicer to my Chief Proofreader: Susan Jasik.

Known problems: MPW

The ".SYM files do not identify tagged CASE variants in records.
MPW Pascal does not put out any type information for objects (MacApp).
MPW C does not put out proper type information for Enums.
MPW C creates unnecessary anonymous types in the ".o" files which the Linker does not remove.
MPW C creates multiple associations for variables that it assigns to registers in a block without specifying the range of the association in the block so that one has to inspect "-Local Vars-" and the object code to decide which assignment is valid.

Known problems: The Debugger

Sometimes *The Debugger* does not properly switch the task name in the menu bar. A work around is to switch to ROM.dsi (task 0) and then back to the window you wanted to bring to the front.

In MultiFinder mode, pressing "option-\" leaves one in the Multifinder task, not the users task. I tried to solve this problem and didn't have much success. The work around is to activate the "-Stack State-" window, highlight a "=nnn" corresponding to a procedure in your task and **Ω-D** to switch tasks or **Cmd-D** to bring up a display of that procedure.

The Debugger is sensitive to the presence of certain INITs and will not start up in background mode if they are present. This is a continuing problem. Known troublemakers are: MacroMaker, Tops, InBox, SuperLaserSpool, etc. You can use a "binary" search to figure out which INIT is causing *The Debugger* a problem, and send me a copy of it so I can fix the problem.

The Debugger does not start at boot time if a Radius Accelerator is installed. This is a known problem, and has defied most of our efforts to fix it. See "Radius Notes" on the White disk.

When The Debugger is loaded at boot time and one or more INITs are loaded after it (zSuitcase), you will get the "ROM patches adjusted" message in The Debugger no matter what you do. Ignore it !!

Oops, The following didn't make it into the Manual:
To pass parameters to a "?UserP", use assignments to global variables in the "action clause" that calls it.

The Scroll bar in the "**Type@nnn**" windows is a "page" type scroll bar that does a page up/down when clicked in the gray region. It is useful in the case you have an array or record. An example is to define the type: `trapTbl = record a,b,c,d:Array[0..3] of ProcPtr end;` and do a `"trapTbl@400".`

Feedback

Your feedback is always appreciated. Many of the features in The Debugger were suggested by users. Remember that I can't fix bugs that you don't tell me about.
If you are having problems with The Debugger, then remember that a test case consists of the failing program and associated files (".map," ".SYM," etc.) along with a description of your configuration which includes: Machine, amount of RAM memory, System level, Ram cache setting, INITs, monitors (size, manufacturer and bits per pixel), etc.

I am in the process of revising my advertising. Any positive or comparative comments that you make (in writing) would be appreciated.

The solution to the problem is to setup a "=type" section in the ".dsi" file as follows.
Cmd-Space and type in **VarElem** to open up a window with the above type definition in it.
Open up a new window, and copy the type definition for **VarElem** from it to the new window.
Fix up the new window so it is now:

```
=Type,REDEF
VarElem = Record  { 202 bytes}
{   0} stPara          : Longint;
{   4} endPara         : Longint;
{   8} isHotLink       : Boolean;
{   9} name            : string[127];
{ 138} format          : string[63];
END;
=END  of the .dsi file
```

and save it as "xxx.dsi". I have boldfaced the changes that I made to the file.
The ",redef" clause on the "=Type" line lets you redefine a type that has already been defined.

The other definition that we worked over was defined as:

```
HotLinkArray = Record  { 210 bytes}
{   0} count           : INTEGER;
{   2} arrayElSize     : INTEGER;
{   4} t               : VarElem;
```

In this case, it was an array of `count` records of `VarElem`. I changed the definition to

```
HotLinkArray = Record  { 210 bytes}
{   0} count           : INTEGER;
{   2} arrayElSize     : INTEGER;
{   4} t               : Array[1..count] of VarElem;
END;
```

and added it to the ".dsi" file.

The point of all this is that The Debugger has, and can display, your type definitions so that you can modify them to optimize the display of values of records.

To view the list of known type definitions that your program has introduced, first make sure that the task indicator (rightmost item) in the menu bar is set to your task, then hold down the shift key and select "**Record/ALL names**" from the **Tables** menu.

Pascal programmers can add the type definitions ("name" equivalences) for HLongint, word, RefNum, etc., to their Interface files, and use these names in their record structures.
While the compiler may not pass them through properly, it is a simple matter for them to copy the record definitions from the source code into a ".dsi" file.

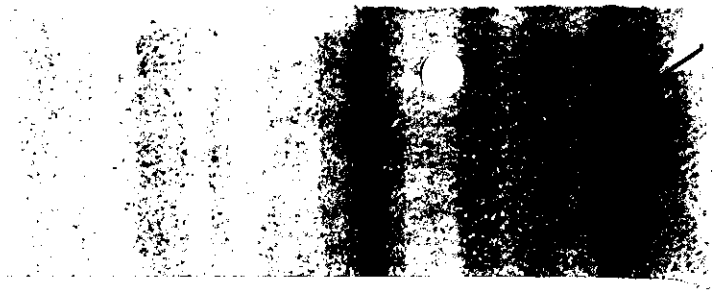
Note that, at present The Debugger does NOT have any facility to process the equivalent of Pascal CONST's (#define in C) so one will have to massage defintions of the form before putting them in the ".dsi" file.

T_arr = ARRAY[0..Max_UpperLim] OF sometype;

For further information on the base types and ".dsi" files, see pages 31 and 33 - 34 of The Debugger manual.

Note that:

Built-in record types (WindowRec, TeRec, etc) take precedence over types defined in the ".SYM" file, and that you may NOT redefine the "built-in" record types.



A Quick Reference for The Debugger V2 - 2/21/89 © Steve Jasik 1989

Installation Procedure

1) Copy the **WHITE** Floppy disk to your Hard disk (and make a backup of the floppy).
   The default ROM snt file setup is for a Mac+ on the "+/SE" and a Mac II on the Univ Disk.
   If your configuration is other than that, then copy the correct ROM snt file into the "Dbgr/Nosy files" folder from the "Mac SE files" or from the blue disk (Univ Version).
2) Copy the files in **"put files in System Folder"** to your System Folder.
3) Run **Nosy**, and create ".snt" files for the **Finder** and **MultiFinder**.
4) If you want to **debug MPW Tools**, then create an ".snt" file for the "MPW Shell" , copy the file "InformDbgr" to to Tools folder in MPW, and add a line to "UserStartup" to execute it.
5) Adjust the names of INITs as per the next section.
6) Boot your machine and let it rip.
7) Optionally you may customize "ROM/CODE.snt" to your system configuration by disassembling "ROM" (in the Nosy/Dbgr files folder) and saving the .snt file.
   If you have any problems with the installation, then see pages 8 - 9 of the manual.

MultiScreen Operation

Mac II, IIx - Debugger takes over the startup Screen. Option-Monitors in Control Panel to set.
Mac +/SE - Only with E-Machines Monitor, install Big Picture™ INIT from Dbgr disk.

RunDbgr versus XDbgr_Startup

You need at least 2 Meg of RAM to run *The Debugger* in the background and 4 Meg or more is preferred to get any real work done. You can **NOT** Launch *The Debugger* after MultiFinder. Mac SE's with third party accelerator cards (Radius, MacPeak): you may have to patch the System to get The Debugger to launch at INIT time. Read the file "Radius Notes" for details.

Macsbug, INITs and Other Gotchas

The Macsbug in "put files in System folder" installs itself in high memory and records patches to the Traps until the Finder is launched. It takes up 8K of memory. To see who patched the Traps, check out the "ROM Patch Info" command in the Display menu. Diehard fans of Apple's Macsbug may rename it "Disassembler" to get it installed.

When *The Debugger* starts up it makes a copy of the "world" (trap vectors, etc.) so that it **will not be affected by patches to the traps** that are set by programs that come after it. To debug INITs that come alphabetically before *xDbgr_Startup* you may rename it or the INIT.
The Debugger is not compatible with all INITs. In particular "SuitcaseII" should be renamed to "zSuitcaseII" so that it loads after *The Debugger*. An alternative is to use Master Juggler which is compatible with *The Debugger* ; it can be used to invoke Desk Accessories inside it. **Note that you use DAs inside *The Debugger* at your own risk!!!**
To avoid interference between the command-option key equivalents in QuicKeys™ and those in *The Debugger*, you may wish to rename QuicKeys™ to zQuicKeys™ .

Menus and Commands, General

Command-option key equivalents show up as **Omega**-Key (Ω) in the Menus.
When the key equivalent is a **lower case letter**, Command-shift-letter will result in a different function (Cmd-B is a simple Bkpt, Cmd-shift-B is a Bkpt with the current action clause attached to it.)
Some unshifted/shifted menu items options are presented in the menu in the format "unshifted/SHIFTED command." For example, in the *Tables* menu, the first menu item is **Record/ALL names**, which lists all the known type names if the shift key is held down, and only the built in Record names if is not.
Menu items that are toggles show up in **Bold face**; a check mark indicates it is selected.

Many commands check for a selection in the front window, and will use it if present.
If nothing is selected, then the command will bring up a simple type in dialog box just below the menu bar. You can **paste into these dialogs** with Cmd-V; and dismiss them with a Cmd-Q. Clicking in the **"Cmd:"** box is equivalent to a return.
Use **Cmd-period** to abort a command or stop continuous Step mode, etc.
The first item in the menu bar is the amount of free space in *The Debugger*'s Heap zone.
The first menu contains a **Help** command.

Windows & Tasks, General

All windows (not dialogs) are Text Edit based (no tabs, 64K size limit). The normal rules for selection (double-click, shift-click, etc.) hold for them. Undo is not implemented.
Use **Option-Click** to obtain a popup menu of the windows to bring one to the front.
In *The Debugger* most of the windows are read only to protect the data from inadvertent modification. You can copy from them.
A quick way of transferring a selection to the "-Notes-" window is to use **Cmd-N**.
Windows that are **task relative** are prefixed with "∂taskname."
A **task** is a process (APPL or executable resource). The list of tasks known to *The Debugger* is displayed in the **"-Task & File Info-"** window (Ω-T) along with the list of open files.
The last item in the Menu bar is the number and name of the task you are in. It will change as you switch from one task relative window to another. Commands that are task relative (such as display Code | Data block) will search tables, etc. relative to that task.

To debug a **task symbolically** you must prepare an **auxiliary file** that contains symbol information that resides in the **same folder from which the task will be executed.**
The aux file may be: a Nosy "snt" file, an MPW or MDS style ".map" file, an MPW V3 **".SYM"** file or the LightspeedC™ PROJ file itself.

The Find command - Searching for Text in a Window

If some object is selected in the front window, then the Find command (Cmd-F) accepts it as the search pattern and looks for the next occurrence of it as a token (separated by delimiters).
If nothing is hilited in the front window, then it brings up the find dialog window to give you more flexibility.

Debugger, Entry/Exit

Cmd-E to exit *The Debugger* to the Problem Program (PP).
To transfer to control from the PP to *The Debugger*, press **Option-** or the Programmers Switch, or use the Programmer's Key INIT.
When in *The Debugger*, use the ~ (tilde) key to view the PP's screen (single screen systems).

On Entry to The Debugger from the PP

On swapping into its world *The Debugger* puts one or more lines into the **"-Notes-"** window to show why you entered it, updates the **"-Registers-"** and **"-Stack State-"** windows, and a window of the procedure containing a disassembly (or source) of the current PC.
The menubar task indicator is set to the task of the procedure containing the PC.
Note: To reset the display of the proc to the current PC, hilite the value of the PC in the **"-Registers-"** window and Cmd-D.

Source Level Debugging (MPW V3 .SYM file or LightspeedC™ project)

For a LightspeedC project, the **source files must be on the same volume** as the "project" file, while with MPW ".SYM" files, the source is assumed to be from where it was built.
Use the "Source only windows" command (Ω-V) to set the default mode of display.
Use a **Command-click** (mouse down) to toggle a window between Source only and source with interspersed assembly. In the source only windows, a"Σ" in column 1 denotes the beginning of a statement. In LightSpeedC **Compile with Debug on** so *The Debugger* has the source line/code info, etc. Running with the LightSpeedC Debugger is optional.

Displaying procedures

Hilite the name of a procedure and **Cmd-D**, or Cmd-D with nothing hilited and you will get a type in dialog box. More frequently in *The Debugger*, you want to look at an active procedure, i.e. one that is in the call chain. Activate the **"-Stack State-"** window, find the procedure you want, hilite the "=nnn" and Cmd-D. Not only is this slightly faster, but it also switches *The Debugger* to the task the procedure is in and positions the display to the point of the call.

Modifying Memory or one of the Registers

Use the Set... (**Cmd-Y**) command and type in Rxi=hexval OR address=hexval/size
where size is **Byte**, **Word** or **Long**.

Misc Notes on the 2/24/90 and later versions

Both Nosy and The Debugger have been fixed to work on the Mac IIci and the Portable.
When you make a ROM.snt file for the Mac IIci or Portable **DO NOT use the old ROM.snt** that is configured for the Mac II. Answer **"n'** to the prompt "read Saved Nosy table (bypass treewalk)".

The behavior of **Cmd-G** was changed as follows:
For Cmd-G the search is case insensitive and ignores token boundaries.
For Cmd-shift-G the search is case sensitive and the search object must be a token.

Note that the default **Cmd-F** search when an item is highlighted in the front window is case sensitive and the search object must be a token.

You may view the values of the Auxiliary (control and MMU registers) by clicking on the "Aux_reg@nnnn" in the "-Registers-" window and Cmd-space.

The behavior of the write statement (Debugger Manual page 38) was changed so that write(var) where var is of type string, Cstring, record or array will print out the the contents of the object in the expected format. Note that you can force a variable or expression to a type in a write statement by using the notation write(VarName:type).

The Behavior of **Cmd-U** was changed as follows:
Cmd-U (Update Globals) and Cmd-shift-U (update Local vars) have been combined into one menu item
A new menu item "Misc:Locals in Current proc" (Ω-U) has been added.

UP your FCB's (File Control Blocks - 1 per open file)
Some of us who are using IBS/PatchLink are running out of FCB's. That is, we have too many open files (error -42). To up the number of FCB's (allocated at boot time by the ROM) you may edit the Boot Blocks of your startup volume in **Fedit** or **Symantic Tools** as follows:

Startup Fedit and select Open volume. Select your startup Volume. Select Edit Boot Blocks from the File menu. Edit the field "Max number of open files". Note that the ROM multiplies the number in that field by 4. That is changing it from 10 to 12 adds 8 extra FCB slots. Click the Update button, quit Fedit and reBoot.

To check the results of your handywork, enter The Debugger, Cmd-W and type-in FcbsPtr.
Select T_FCBSptr@nnn and look at the resulting window. The value of "lenFCB_buf " divided by 94 is the number of FCB entries (Cmd-[and typein "lenFCB_buf" / #94).

MPW C Programs - Important Note
The Debugger automatically translates the type **Array of Unsigned char** into type string, and displays variables as such. **Arrays of Char** translate to type SignedByte on the Mac are NOT given the same special treatment.

LS Fortran Notes
This version of The Debugger supports LS Fortran V2 common variables and Arrays. You will also need the MPW 3.1 Linker to create the ".SYM" files.

Cheap Memory
The Chip Merchant in San Diego, CA has the best prices for 1 Meg SIMMs. As of 2/26/90 the price was $63 a meg for 80 ns Fast page mode SIMMs (the kind that the IIci needs). Their phone numbers are 800-426-6375 outside of CA and 619-268-4774 inside CA or international. An Alternate source which takes credit cards is Peripheral Outlet in Ada, OK. Their phone numbers are: 800-332-6581 and 405-332-6581.

Nosy and mis-segmented procedures
To view a list of the procedures that are referenced only once and in a different segment than the caller do the following: After a Treewalk, press Cmd-Y to enter TTY mode, then press M and return to go to the Maps menu. Enter "oxx" and return to open a file. Enter "s1" and return.
When the list is finished enter "o" return, return, "d" return. Then open the file "xx".
MacApp users - The map will contain spurrious entries for method names, ignore them.

Breakpoi··

To set or ···r a breakpoint, hilite a line in a procedure window and **Cmd-B**.
To attach the current "action clause" to a breakpoint, **Cmd-shift-B**.
To define an "action clause", hilite it and **Ω-B**. It will become the current "action clause".
To change the current "action clause", de-hilite any selection in the front window and **Ω-B**.
Then enter the number of the "action clause" you wish to make current.
To delete an "action clause". de-hilite any selection in the front window and **Ω-shift-B**.
Then enter the number of the "action clause" you wish to delete.
Do not set a breakpoint in ROM patch code that The Debugger may execute!

Controlled execution (stepping)

You may **single step by Cmd-comma** (Step Through) or Cmd-? (Step Into) or you may **step** continuous by Cmd-semicolon (Step Continuous). To **bypass the Step continuous dialog**, use **Cmd-shift-semicolon**. When in continuous step mode, hold down the Cmd key to **pause** execution and Cmd-period to stop it. If you step into a procedure that you don't want to be in, use Cmd-quote (Go Until Caller) or **Ω-R** (Go Until Exit ROM) to get out of it.
To paraphrase Charles Dickens: Tis a far, far better thing that you **step continuous**, then trying to single step and lift your fingers off the Cmd and comma or "?" keys every time.

Changing the Program Counter (PC)

To change the Program Counter (PC), hilite a line containing the new PC, and use the Set PC (**Cmd-P**) or Go Until PC (**Ω-P**) command. You do this in **Source only windows at your own risk**, as the compilers may have generated code that initializes registers in non obvious places.

Displaying memory, values of variables, Hypertext

Hilite an address in any window and **Cmd-space** to obtain a Hex/Ascii display of memory.
To display memory according to the Type "TTT", hilite an expression of the form:
"TTT@nnn" and Cmd-space. nnn may be any expression.
Use **Cmd-8** to toggle a Cmd-space window between one that is static and one that is updated on each entry to *The Debugger*.
To display the **definition of a Type**, Cmd-space and enter the Typename.
To display the **value of a global variable**, switch to the task containing the global, hilite the name, and **Cmd-W**. The name, its value, etc. will appear in the "**-Values-**" window.
To **typecast** a name, **Cmd-W** and enter "name:Type"
To **delete a name** from the values window, hilite the name and **Cmd-shift-W**.
To view the values of the **Local variables** of the active procedures, **Cmd-shift-U**.

Watching memory for changes

First determine the hexadecimal address of the variable that you want to have watched from an "asm" window or placing the variable in the values display, then **Ω-W**, and enter the address of the variable and the number of bytes that you want to watch. On pressing return, *The Debugger* will exit to the problem program. The manual describes other variations that are possible, such as watching a section of memory within a heap block, etc.

Intercepting Traps

You may set a trap intercept by hiliting the name of a trap and **Cmd-J**, or Cmd-J and type its name, or **Cmd-shift-J** for a fancy dialog to select trap names by suite or name.
To **clear an intercept**, Hilite the trap name or line containing it and **Ω-J**.

Trap Discipline and Handle Zapping - the Bondage Menu

Trap discipline checks the arguments to a mac trap call on entry for consistency, and breaks into *The Debugger* with a **message in the "-Notes-" window** if they are incorrect.
Cmd-5 (Ignore Current Error) will suppress future Discipline messages for the Trap call at PC.
Handle Zapping is really a sub menu item of discipline. When it is selected, blocks in the heap will be set to a value that should cause an address error if the block is referenced after it is released (_DisposPtr, _DisposeHandle). See the manual for more details.

Heap Display & the Where the F... is it Command

Cmd-H to bring up a display of the Heap zone of the current task. **Cmd-shi··** ·H to display the System Heap zone. To find out what heap zone and block an address is in ···te it and **Ω-G**.

The Expression Language

A language based on the Pascal Syntax, it contains arithmetic operators, references to registers (Rxi), user variables (name or ∂taskname.Varname), constants (default is hex, #nn for decimal base), debugger variables and functions (prefixed with a "?"), and for the curious, who insist on knowing where their procedures are in memory, the expression "@procname" will return the address of loaded procedure.
For a full description and examples consult the Help file or the manual.

".dsi" Files and Debugger Flags

".dsi" files are documented in the file Tutorial.dsi and the Manual, check it out.
Debugger flags are described in the "ROM.dsi" file. **Ω-Z** to see their current values.
To **change a flag value**, select lines of the form "=Flag" , "flag_name = 0 or 1;" and **Ω-F**.

Debugging MPW Tools

Is automatic if you have followed step 4 of the installation procedure and added a line to your UserStartUp" file to execute "InformDbgr" once per Launch of the MPW Shell.

Debugging INITs

You debug an INIT's setup code by supplying an auxiliary file. To symbolically debug the Traps that the INIT patches, rebuild the ROM "snt" file with the INIT installed.

Debugging Programs that jump off into the wild blue yonder

If you have a computer with an 020/030 CPU then turn on **"Trace Jumps"** in the **Go"** menu and rerun the program. If you don't then go out and borrow, buy or steal one.

Debugging Programs that blow off in the ROM

Step back from the computer, take a deep breath and count to ten. Find a Voodoo doll that looks like an Apple ROM programmer and stick some pins into it. If it is past 1 AM then go directly to bed, do not pass GO. More seriously if the problem is repeatable, as most bugs are, then rerun the program with Discipline on, and if you have a computer with an 020/030 CPU then turn on **"Trace Jumps"** in the **Go"** menu. If you are not me, then avoid trying to step into the ROM code as you will just get lost in the Rats nest that it has become. Look at the code in your program, read the accumulated tech notes, Inside Macintosh Volumes 1 through 5, etc until you can pin the blame upon your program. All else failing, call Mac DTS or John-Louis Gasse and tell them what you think of their @#$!& system.

Terminating the current debugging session

When the sushi has hit the fan or the magic Bomb box has appeared, you should consider the "Boot System" or "Flush System and Boot" commands as viable alternatives to continuing the current masochism. If you don't want the Volumes flushed on a Mac II/IIx, then use the Reset switch on the side of the Machine.

Terminating an Application (Multifinder mode only)

To do this the PC must be within your APPLications heap zone, so adjust it first if necessary, then select the Shutdown APPL command from the first menu.

Shutting down the Debugger (RunDbgr mode only)

This operation can be done with relative impunity in almost all cases.
Use the "Shutdown Debugger " command from the Files menu to terminate your application and *The Debugger,* and transfer to the Finder, MPW or LightSpeedC.

A last Word - don't say I didn't tell ya so

Make a copy of your System file on your hard disk, and keep a Bootable floppy disk with some recovery utilities handy. MPW's *Rezdet* will detect corrupted System files, etc.

Jasik Designs

343 Trenton Way Menlo Park, CA 94025 (415) 322-1386

Nov 7, 1988

Dear Mac II Nosy/Debugger user,

Welcome to the world of Nosy/Debugger . Enclosed you should find a 800K floppy disk, a Delphi™ authorization sheet, and a copy of the Nosy and Debugger manuals. I hope you will give it an evening of active reading at your Macintosh.

If you have a modem, you may use the authorization label on the enclosed sheet to join Delphi. In addition to the MacNosy SIG, you may wish to join the Macintosh SIG, ICONtact™, which contains an extensive collection of public domain utilities, Sunday night conferences, etc. I am also active on Compuserve.

Please note that for owners of the +/SE version there is no formal warranty registration. Because of this, **you must include your labeled Debugger disk** with any correspondence to me in order to obtain updates or upgrades.

The Mac +/SE versions of Nosy/Debugger run with System 5 or 6.x, and are compatable with 68020 accelerator boards. A version of Nosy and The Debugger that also runs on the Mac II and IIx is available. You may upgrade from the Mac +/SE version to the Mac II (Universal) version of Nosy/Debugger by sending me your Labeled Debugger disk and $150.

Many of you use *Nosy* and *The Debugger* as tools to learn about the Macintosh innards. While the documentation does justice to them, it is not a substitute for the variety of texts available on 68000 assembly language programming, and the Macintosh OS. The section "Alternate Sources of Truth" in the Nosy manual lists of number of recommendations. One of those is MacTutor magazine. In addition to being an excellent source of information and gossip, it is my prime advertising medium. In it, you will find articles about "The Debugger" (April 87) and news about future versions of Nosy.

Nosy and *The Debugger* are continually changing products. Among the new features is a Trap Patch Recorder that is named *MacsBug*. When it is installed in the System folder, it will patch _SetTrapAddress and record all patches to the System until the Finder starts. The format of the tables built by *Macsbug* are defined in the file "TPR.defs". If for some reason you want to install Apple's Macsbug, then you may rename it *Disassembler*. Both *Nosy* and *The Debugger* use the results of *Macsbug* to perform their functions. When *Macsbug* (aka TPR) is installed it uses 5K.

Nosy uses the table to give the patches to the traps mnemonic names. If, for example, _Open is patched by the system file and an INIT, then the patch to _Open will be named _Open or PFn_Open, the System patch "SyP_Open" and the original code in ROM, "ROM_Open". The "ROM Patch Info" command in the Display menu will show you a complete list of the patches and the associated procedure names.

Happy sniffing,

Steve Jasik

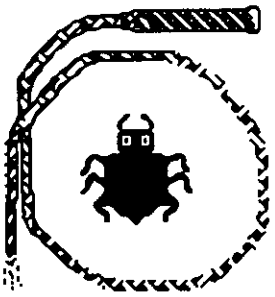
Steve Jasik **Compuserve ID: 76004,2067** **Delphi: "MacNosy"**

Information

Debugger Tech Note #2

Debugging MacApp Programs

Sept 89



Control

This Tech Note is for programmers who use MacApp 2.0ß9 with MPW 3.1ß or later.

Recent additions to The Debugger for MacApp users include:
The ability to step into (Cmd-?) MacApp method calls at the source level
The addition of an "Object Inspector"
The ability to do writeLn's to The Debugger's "-Notes-" window from a MacApp program.

The Object Inspector commands in The Debugger are in the "TaskName" menu.
The menu items will become active for MacApp programs which are built with the special MacApp library routines after the MacApp initialization is called. The changed versions of the files should be copied to the appropriate folders and the applications rebuilt.

The changed files are:
IN {MacApp}Libraries: UObject.a, UObject.globals.p and UObject.TObject.p
IN {MacApp}Interfaces:Pinterfaces: UObject.p

ebuild any applications you want to try out with :

MABuild Applname -LinkMap -SYM -AutoBuild -NoDebug

Note: If you are using IBS then you may omit '-LinkMap -SYM' which are set by IBS.

The program will collect information about Objects when The Debugger is running and the ".SYM" file is in the same folder as the application.

The following capabilities of the MacApp Debugger have not yet been moved to The Debugger:

The MacApp Performance analyzer.
Miscellaneous assertions (error checks) that are activated by building the program with the Debug flag set such as the Focus Assertions, etc.

As an alternative to the Recent History command in the MacApp Debugger see the description of the "Proc Entry/Exit Trace" (Stops Menu) command in the Help file (Help Command).

The Object Inspector Menu (TaskName) in The Debugger contains the following items:

Class Tree Of ... - A hierarchal display of the object Classes. Hilight a class name to view the subtree of a class. The Hex number to the left of the class name is the internal ClassID.

Objects by Time - A list of the known objects in chronological order. Each line is of the form TTT_@nnn . Click on it and Cmd-space to see the value of the object.
Hold down the shift key to create a new window.

Objects by name - A list of the Known objects sorted by name.
Hold down the shift key to create a new window.

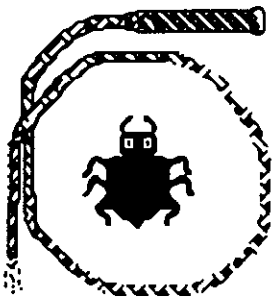
Objects of ... - Highlight or type in the name of a Class to obtain a window with the objects Of that class.

Methods of ... - Highlight or type in the name of a Class to obtain a window with the methods Of that class.

Debugger Tech Note #3

".dsi" files Revisited

Oct 89



Information

Control

A number of miscellaneous issues have come up recently in the process of answering tech support calls. Many of the questions/problems can be solved by using the ".dsi" file and the **Tables:execute selection** command.

The structure and commands in a ".dsi" file are described on pgs 33 - 34 of The Debugger manual.

=Source - setting the source Paths for a program with a ".SYM" file

The implementation of this command has been changed to allow for multiple source paths. The correct way to specify multiple source paths is:

```
=source
path_name1:       (quotes are optional, path_names must start in column 1)
path_name2:       final colon is a must
etc
```

All source paths must be specified at one time. The effect of the =Source command is to first clear the (source path) table, and then to add the specified source paths.

versions of The Debugger prior to 10/4/89 did not handle the specification of source paths on volumes other then the boot volume correctly,

=Flags - Toggling Debugger options

The default **ROM.dsi** file contains a complete listing of all The Debugger options that may be set, and a somewhat cryptic explanation of what each of them controls.
One may view the current settings of the flag values by executing the **Misc:Sts Dump** command.
The format of a statement in the flags section is `flagName = val;` where val is 0 or 1.
Some of the flags that are of interest to you are:

Inst1_Bkpt - if ON, then The Debugger will setup a courtesy breakpoint at the first instruction of all Applications, Tools or Code resources that are being debugged.

Bad_Zero - If ON, then The Debugger will put a bad value in location zero of RAM memory so as to catch programs that dereference NIL handles. The value it puts in zero is `$FFFDFFFD` which will cause an odd address exception on 68000 machines and a bus error on 68020/030's.

If you get the message "Dereferencing Zero is a No No", then you will also get a window titled T_BCFSF on 68020/030 machines. BCFSF stands for Bus Control Fault Stack Frame which is described in the Exception processing chapter of the 68020/030 Motorola manual.

The default setting of Bad_Zero is on, as I and you do not expect good debugged Macintosh programs to dereference zero. Unfortunately there are a couple of bad Apples in every barrel, and 32-bit QuickDraw is one of the offenders.

The action you should take is to enter, select and execute (Ω-F):

```
=f
bad_zero = 0;
```

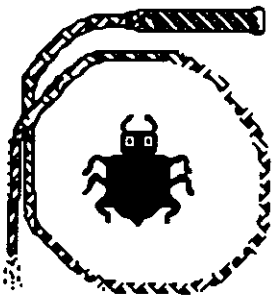
and to clear location zero with a Set ... (Cmd-Y) and 0=0 entry.

Information

Debugger Tech Note #3

".dsi" files Revisited

Oct 89, revised Dec 89



Control

A number of miscellaneous issues have come up recently in the process of answering tech support calls. Many of the questions/problems can be solved by using the ".dsi" file and the **Tables:execute selection** command.

The structure and commands in a ".dsi" file are described on pgs 33 - 34 of The Debugger manual.

=Source - setting the source Paths for a program with a ".SYM" file

The implementation of this command has been changed to allow for multiple source paths. The correct way to specify multiple source paths is:

```
=source
path_name1:       (quotes are optional, path_names must start in column 1)
path_name2:       final colon is a must
etc
```

All source paths must be specified at one time. The effect of the =Source command is to first clear the (source path) table, and then to add the specified source paths.

rsions of The Debugger prior to 10/4/89 did not handle the specification of source paths on volumes other then the boot volume correctly,

=Flags - Toggling Debugger options

The default **ROM.dsi** file contains a complete listing of all The Debugger options that may be set, and a somewhat cryptic explanation of what each of them controls.
One may view the current settings of the flag values by executing the **Misc:Sts Dump** command.
One may **toggle the value of a flag by hilighting it and pressing Ω-F.**
The format of a statement in the flags section is `flagName = val;` where val is 0 or 1.
Some of the flags that are of interest to you are:

Inst1_Bkpt - if ON, then The Debugger will setup a courtesy breakpoint at the first instruction of all Applications, Tools or Code resources that are being debugged.

Bad_Zero - If ON, then The Debugger will put a bad value in location zero of RAM memory so as to catch programs that dereference NIL handles. The value it puts in zero is `$FFFDFFFD` which will cause an odd address exception on 68000 machines and a bus error on 68020/030's.

If you get the message "Dereferencing Zero is a No No", then you will also get a window titled T_BCFSF on 68020/030 machines. BCFSF stands for Bus Control Fault Stack Frame which is described in the Exception processing chapter of the 68020/030 Motorola manual.

The default setting of Bad_Zero is on, as I and you do not expect good debugged Macintosh programs to dereference zero. Unfortunately there are a couple of bad Apples in every barrel, and 32-bit QuickDraw is one of the offenders.

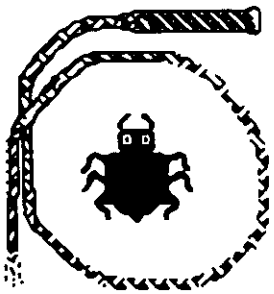
The action you should take is to press **Ω-Z** to bring up the Debugger Sts window, hilight bad_zero, a press Ω-F to toggle the value of the flag,
and to clear location zero with a Set ... (Cmd-Y) and 0=0 entry.

Information

Debugger Tech Note #5

Debugging INITs and Executable Resources

November 89



Control

The procedures for debugging INITs and executable resources (DRVRs, CDEFs, ...) have changed a bit since the manual was written. To keep you from being burnt by some of the gotchas, I have collected the pertinent material in this tech note.

To Debug INITs:

1) The INIT must load after The Debugger for you to debug it, so prefix it with a "z" or rename xDbgr_Startup.

2) Create aux files as described in step 3 below.

3) For debugging purposes, keep in mind that The Debugger associates only one address with a resource or INIT. Debugging the INIT symbolically is easy, but what most of us want to debug are the patches to the system that the INIT makes.
Most INITs are loaded into an area of memory will become the Application heap, and then the INIT moves the patch code into the System heap or the area above BufPtr.
If you wish to debug the patches, then omit the code that moves the patches, and load the INIT directly into the System Heap by setting the System Heap and Locked attributes for your INIT in the MPW Linker (-ra =SysHeap,Locked) or Rez or ResEdit.

To Debug executable resources:

0) Think C users will have to build Application with Macsbug names and run it through Nosy to create an ".snt" file. Use MPW if you want source level debugging.

1) To debug any resource, change ROM.dsi so that `Dbg_Rsrcs = 1;` in the `=Flags` section.

2) MPW users: To build DRVRs incorporate DRVRHdr.a into your make.

3) See page 32 of The Debugger manual for info on aux file naming conventions (.SYM, .dsi, ...). If you need an example of what you should name your aux files, use Nosy to disassemble your "CODE" resource, and create an ".snt" file (Misc:Save snt, reRead .aci command in window mode). If you are using a ".SYM" file, then remember to trash (or rename) the ".snt" file after you have used it as ".snt" file takes precedence over the ".SYM" file in The Debugger.

For MPW users, your script should look like:

```
LINK -o myRsrc -SYM ON -rt RSRC=$hhh ...      # where $hhh is the Rsrc's ID in hex
REZ  … -o RsrcFile              # myRsrc -> RsrcFile
setFile myRsrc.SYM - m .        # touch it so it has a later mod date then then RsrcFile
ReName  myRsrc.SYM  'RsrcFile/RSRC_hhh.SYM'  # note the single quotes around the name
```

4) If your resource type "RSRC" is not one of the standard types listed in the Debugger's table of executable resources (page 30), then you need to add the following lines to a ".dsi" file that is **executed before** your resource is loaded:

```
=G        ; specify Generic resource type to be debugged
RSRC
```

or you may type the lines into the "-Notes-" window, and execute them, etc.

5) See Tech Note #3 for info on creating a ".dsi" file.

6) Multi-segment DA's: create a ".SYM" or ".snt" and a ".dsi" for each segment you want to debug.

Authorization Label:

Code above is worth $29.95

Six Simple Steps to Register your Complimentary Delphi Membership

1: Obtain a Phone Number

If you live in the Boston area, use the Delphi number (617-576-0862).
Outside the Boston area, dial Telenet (800-336-0437) or Tymenet (800-336-0149) for a local phone number.
International users should contact their local P.T. & T. for connect information to Telenet.
Canadian users may access Delphi via Datapac or INET 2000 (Telenet & Tymenet is available in some cities).

2: Initial logon procedure

With your computer and modem, dial the number you obtained in Step 1.
For Telenet, type **return** until (2 or more times) you get an "at sign" prompt (@). Then type "**c delphi**" **return** .
All users: in reply to the prompt "**username**", type in the username on the above label and press **return**.
You will be asked to enter your authorization code exactly as it appears on the above label.
Do so, then press **return**.
You are now online!

3: Warranty Registration

You will be asked for information about the product you purchased for warrenty and registration records.
Follow the online instructions, making sure to press **return** after every answer.

4: Delphi Demonstration

After completing your warranty registration, you'll be given a brief demonstration of **Delphi**. You'll have an opportunity to see why users rank **Delphi** number one in overall satisfaction (Datapro Research Corp, July 84).

5: Signup Procedure

After the Delphi Demo, you will be given the opportunity to register your Delphi account, free of charge. Your registration entitles you to a lifetime Delphi membership, and your first evening hour of use at no charge.
Simply follow the online instructions to complete the membership application.
The information you supply as part of your registration must be complete and correct, so that Delphi's customer service can process your account registration and provide you with the excellent support and service you expect and deserve. Please be sure to include the phone numbers where you can be reached if Delphi customer service has any questions about your account.
While Online, you will receive your initial password and be advised as to when you can activate your account.
Please remember that the password is temporary and will be changed by you, after you log on for the first time.

6: Logging on for the first time on your account

Follow the same procedure as in Step 2 up to the point where you are asked for your username.
When prompted for "username" – type the Delphi name you have chosen.
In reply to the prompt "PASSWORD" – type in your temporary password followed by a return. It will not be echoed by Delphi for security reasons.
Welcome to Delphi!
Our guided tour director, Max, will show you how to change your password and get started.

Entering the MacNosy SIG

Shortly after you have registered, your name will be forwarded to Jasik Designs. We will then enter you in the MacNosy SIG, and send you an Email message to that effect, along with directions to the SIG

Thanks for joining Delphi, it's nice to have you as a member!! Our Customer service number is 1-800-544-4005

As of 12/23/86 Delphi charges are $7.20 an hour for evening time (6 PM to 7 AM) and $17.40 for daytime. Connect charges are independent of the Baud rate.

16/11/89

Nosy/Debugger Registration for Mac II Version

In order to obtain updates by mail as they are issued, please affix a **typed** copy of your **name** and **mailing address** to this form and return it to:
(forms returned **without the name of a person will be thrown away)**

Jasik Designs - Mac II Register
343 Trenton Way
Menlo Park, CA 94025

Thank You

SN# 10/22 - 8

ComputerWare **10/22/89** **IBS V1/ IIci**

Steve Jasik

This slip good for Debugger Ver 2 Updates in 12/89

Feedback:

What is your configuration?

CPU SE-30 RAM: 4M Hard Disks 40 M

Monitors: External Apple 13" Color Special Cards, etc Super Mac Technology color card

What language development Systems (MPW, Think C,etc) do you use?

MPW

Do you use MacApp ? yes Do you intend to use C++ when available ? yes

What kind of problems did you encounter when you tried to install The Debugger and IBS?

no problem

Suggestions for features, improvements, etc: not yet

ANDREA CARPENE'
c/o E.N.E.A.
VIA MARTIRI DI MONTESOLE, 4
40129 BOLOGNA
(ITALY)

Installation Procedure

Install the latest version of The Debugger (dated 9/24/89 or later) and the associated aux files in the Dbgr/Nosy files folder.

Save away the files in the MacApp library which are being replaced by my versions.

Copy the files in SIJ_MacApp_Libs to the appropriate folders in your MacApp folder.

Paste the contents of the text file **"TList_Redef"** into the ".dsi" file for any MacApp applications you are going to build.

Rebuild any applications with:

```
MABuild  Applname -LinkMap  -SYM  -AutoBuild -NoDebug
```

Misc Notes

To set breakpoints on (all?) the methods of a class:
Use the Methods of command to bring up a list of the methods, Cmd-A to select all, and Cmd-B to set breakpoints at the entry to the procedures.

The new code that I have introduced into MacApp is conditionally assembled in by the setting of the flag "qExternalDbgr" which is set to TRUE in Uobject.p. Eventually this MAY be made into a MABuild option.

Use the compare Tool to see what changes I have made to the files.

Errata for version 1.1 of **Nosy Print**
July 10, 1986

Nosy Print is a desk accessory that compensates for Mac Nosy™'s lack of printing facilities by reaching into Nosy's data structures and printing the topmost Mac Nosy window on the ImageWriter. (It will probably print on the LaserWriter as well, though that's not yet been tested). **Nosy Print**'s use of Nosy's data structures (for Mac Nosy 2.0 and greater) has Steve Jasik's blessing.

Nosy Print is intended to be installed directly into Mac Nosy. To do so, launch the Font/DA Mover, and then hold down the option key when clicking on **OPEN**. This brings up the standard file dialog with all applications listed. Select Nosy (or a copy of Nosy, if you're paranoid) and move **Nosy Print** into it. (I usually reboot at this point, out of latent distrust about what Font/DA Mover may be leaving in memory.)

When selected from the Apple menu, **Nosy Print** first puts up an "About" dialog, then the standard "Page Setup.." dialog, and finally the "Print.." dialog. (Subsequent selections of **Nosy Print** will only put up the "Print.." dialog.) The contents of the topmost Nosy window should then start printing.

There are a few minor problems with version 1.0 that I intend to deal with in a later release (unless Steve puts printing code into Nosy himself by then). Long lines are truncated in printing. This hasn't been a problem for me, as the long lines are typically trap commentary added by Nosy. If you run into other problems, please drop me a note.

Nosy Print works fine on my configuration, which consists of a Mac+ and a DataFrame 20, but I can't guarantee that it will work on all other configurations. If you have a problem, drop me a card describing it, and I'll see what I can do.

Nosy Print is written entirely in LightspeedC™, with very little trickery.

David W. Smith
P.O.B. 360311
Milpitas, CA 95035
USENET: well!smith

P.S. If you're really tight on disk space, the PICT resource used for the "About" dialog can safely be spliced out using ResEdit. Please don't pass on lobotomized copies without this note, so that people will know who to contact if they have problems.

P.P.S. Shareware authors deserve your financial support. This is free however, so don't send money.

Misc Notes on the 10/22 and later versions

Both Nosy and The Debugger have been fixed to work on the Mac IIci.
They do not work on the portable.

The behavior of **Cmd-G** was changed as follows:
For Cmd-G the search is case insensitive and ignores token boundaries.
For Cmd-shift-G the search is case sensitive and the search object must be a token.

Note that the default **Cmd-F** search when an item is highlighted in the front window is casesensitive and the search object must be a token.

The Debugger and Nosy now handle '.SYM" files with more than 2047 user types.
This was a problem with large C programs.

The behavior of the write statement (Debugger Manual page 38) was changed so that write(var) where var is of type string, Cstring, record or array will print out the the contents of the object in the expected format.
Note that you can force a variable or expression to a type in a write statement by typecasting it (write(mytype(var))) or by using the notation write(VarName:type).

Nosy Usage Notes and Documentation

Revised 1/16/86

Copyright Steve Jasik 1986

My address/ phone is: Steve Jasik
343 Trenton Way
Menlo Park, Ca. 94025
Phone: 415-322-1386

In the UK: MacSerious Software
36 Queen Street
Helensburgh G84 9PU
Phone: 0436-6971

See the file "Addenda" on the Nosy disk for a list of new features and bug fixs.
Disassembling Rom under MacWorks disassembles the file "sij/ramrom",
create it on the Mac with a standard system (no Hyperdrive, etc), not the Lisa (Mac XL) !!
Online Help is available, type "? for info about the current menu and "?X" for info about command "X".

Copyright Notices

Acknowledgments

Many people helped me develop Nosy and deserve credit. Thanks go to:

Larry Rhyne *for letting me use his Lisa until I could afford my own.*
Dennis Brothers *for helping me spread the word on CompuServe.*
The many members of the press who helped me get the word out.
John Mitchell *(Fedit) who provided extensive testing and encouragement.*
Wayne Martin, *who reached into his pocket and found some money to keep me going.*

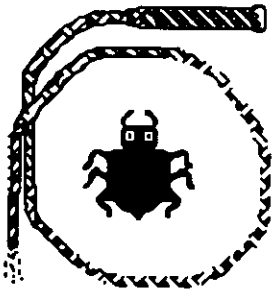
64K ROM vs 128K ROM

Nosy V2 is being released to support the 128K ROM, and will not disassemble the 64K ROM (it will give you a message, and revert back to the file select box if your machine still has the 64K version installed), I suggest that you consider upgrading it, as there are some noticable performance enhancements in QuickDraw, as well as providing the support code for the double density disk drives and new hierarchical file system. On a Mac with the 64K ROM's installed Nosy will function correctly disassembling programs other than the installed ROM.

Debugger Tech Note #4

An Interfile Marker Facility for MPW 3 Users (aka "CTags")

November 89



Information

Control

The mark facility has been with us since MPW 2.0, and can be used as an aid to navigating one's way around the source code in large programs. The basic mark facility works within a file. I have built a set of MPW tools and scripts that lets one extend it to an arbitrary set of files.

To obtain the ability to move around in one's source code easily, one needs the following: A tool to mark the source files, a tool to build the database of what file a procedure is in, and a tool to search the database and issue Shell commands that will open and position the file.

MarkTool is used in conjunction with the script, **MarkScrpt**, to mark all the source files in a given program. In order to avoid having to write a parser for each language, MarkTool reads the object (.o) files that have been created by the compilers with the "-SYM ON" option, and it produces a text file of Shell commands to be executed that creates the markers. You will have to mark ".a" files manually if procedures/ functions do not begin with PROC/FUNC directives.

The call to MarkTool is:

```
MarkTool file1.o file2.o .... >Output
```

To create a command file for object files in a directory "dirname", and to mark the source files:

```
MarkTool dirname:≈.c >mark_commands_file  # where ≈ is option-X
# if MacApp then open mark_commands_file and change Underscores to periods ( _ -> .)
Execute mark_commands_file                # generate the markers
```

The output of MarkTool will look as follows:

```
Directory `GetFileName -d`                # set directory to that of the source files
Target Scan_Mod.p                         # open the sourceFile as the "Target" window
UNMark `markers "{target}"` "{target}"    # remove OLD Markers, will FAIL if none
MarkScrpt Scan_Mod.p 5760 OBJ_FMTERROR    # mark the file
  ...
MarkScrpt Scan_Mod.p 10978 SCANOBJ_FILE
Close -y "{target}"                       # close the window
```

The next step is to use **Mark_file** tool to build or rebuild the database of markers.
The call to Mark_file is:

```
Mark_file "{IBS_Project}" ∂
SourcePath1: SourceFileA1.p SourceFileB1.p  ... SourceFileZ1.p ∂
    ...
SourcePath2: SourceFileA2.p SourceFileB2.p  ... SourceFileZ2.p
```

Note that you must include ALL files in ALL directories that are in your program and contain the body of procedures in this command. **DO NOT** include header and Interface files in it.
Use the **make_file_List** script to form the list of files in your project and execute its output That is, you first set IBS_Project, and then execute make_file_List.

The UserStartup•IBS file contains the **"GOTO Def of ..."** menu item (Cmd-J).
You can now select (hilight) a procedure or function name, and Cmd-J will take you to its definition, or type Cmd-J and enter the name of a procedure.
At present this scheme is of limited use for OOP programs since most of the method calls do not include the class name. You will have to copy the method name, Cmd-J, type "ClassName." and paste the method name in.

Files on the MacNosy disk (See the Get Info boxes in the Finder)

* **Nosy** - the disassembler (Mac/MacWorks version)
* **Launch Nosy** - a dummy document to be left on the main desktop to launch Nosy in a HFS system
* **"ROM"** - A dummy file needed by Nosy to select ROM on the Mac/Macworks.
* **sij/rom.c, sij/absval.c, sij/syserr.c** - condensed symbol table files
* **sij/disatbl** - tables for the disassembler
* **sij/nhelp.text, -Help-** - help files (text format)

Copyrom - a program that copies the first $4E00 bytes of ram and all of rom to the file "ramrom" (needed to disassemble ROM on the Mac XL.)
PrSpool - a simple print spooler that prints files on the Imagewriter in its native font.
Prspool.text - source of the print spooler.
Addenda - summary of bug fixs and new features added to Nosy since release.
AMACS - defines the macros output by Nosy (MDS format "include" file)
Heapmgr - Pascal source for the table manager I use
Getfile - Pascal source for the call to SfGetFile. Check it to see my criteria for disassembling a file.
Steal that Sort - Jrnl file to extract the fast hand coded Shell sorts from Nosy.
Window, Window.map - example that illustrates the use of ".MAP" file creation.
MacLibglue, HFS Libglue, Test_Nosy - example programs and jrnls

NOSY WILL NOT RUN PROPERLY IF YOU RENAME THE FILES PREFIXED BY AN '*'

If you find trash files of the form 'Tgggg' where g is a garbage char, then delete them. They appear because Nosy crashed or you hit the reset button when Nosy was running.

To run Nosy on the **Mac** : ***** MAKE A COPY OF THE ORIGINAL DISK *****

A **minimal** working copy of the Nosy disk consists of only the file icons on the top line in the Finder (Nosy, Nosy files, System).
When running Nosy from a hard disk with HFS, move Nosy to the "Nosy files" folder .

To run Nosy on **MacWorks** : ***** MAKE A COPY OF THE ORIGINAL DISK *****

Move "copyROM" to a empty disk and run it on a Mac, not a Mac/XL
(without Macsbug).
Copy the file "ramrom" over to the Hard disk as "sij/ramrom".
Copy the rest of the Nosy disk to the Hard disk. You will not need copyrom, etc.

You can use MacNosy to disassemble; ROM, a Mac resource file, Macsbug, or the resource fork of the System file (Pack,DRVR, etc). To Disassemble Macsbug call it 'xmacsbug' and change the name of the .jrnl file to 'xmacsbug.jrnl'. Nosy special cases files containing the string 'macsbug' (jrnl file for Macsbug on Delphi).

The files in the examples folder illustrate the use of Nosy, and were put on the disk as samples, so give them a try. Use the ROM "jrnl" file to learn about block splitting.

User Interface and Command syntax

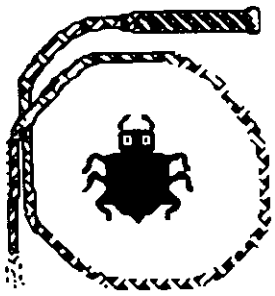
Nosy uses a simple scrolling window that fills the entire screen to display the output of commands and echo keyboard input from the user. The rest of the interface was borrowed from the Lisa Workshop. Menus and queries are displayed on the current line.

Queries follow the Workshop conventions, i.e. the string in brackets ([ ]) is the default reply that you get with a carriage return. Note that the obvious abbreviations are y = 'YES" and n = "NO".

Debugger Tech Note #6

Change History - 3/89 to 3/90

March 90



Information

Control

A **summary of external changes** to The Debugger that were made after version 2 of the manual was printed. Changes discussed in other tech notes are not mentioned here.

- Handle C++ programs, "demangling" the names & handling names of the form "class::method".
  Current release demangles the constructor/destructor to "class::__ct" and "class::__dt".
  There are no special provisions for overloaded names.
- Add source level stepping for programs with .SYM files.
- Change single screen mode to Swap Screens versus Split Screen modes.
- In swap screen mode, allow either screen to be the main one (new version of xDbgr_Startup).
- Extend the type compiler to handle enumerated types, integer subranges, Pascal OBJECTs,etc.
  See the file "AppleTalk_EQUs" for examples.
  typName = OBJECT(className) fieldList Methods END; is the syntax for Objects.
- Allow the redefinition of types introduced in a .SYM file via **=TYPE,redef** in the ".dsi" file.
  Redefine your types to improve the display of them when a compiler "mangles" their definitions.
- Allow variable upper bound in last array of a record definition (Do a Cmd-space of TeRecord, etc).
- Fixed is now a base type that displays a longint as a flting pt number (16/integer_part,16/fraction)
- Expressions Parser - integers appearing in type declarations default to decimal base unless explicitly preceded by a '$". The "$" is NOT persistent as it is in arithmetic expressions.
- Record display - Add a kludge to auto recognize Color Grafports and new style TeRecords.
- Add "Dbg_Rsrcs" flag in .dsi files to turn off the recording/debugging of resources.
  This was done so as to minimize screen flashing when the user has only one screen.
- Add the "flag" BAD_ZERO to control the "garbaging" of Loc zero.
- Cmd-W of a local var name will bring up the "Local Vars-" window & highlight the name.
- Allow local and global variable Value windows to be dynamic (**Cmd-8**).
- **Changes to Both MacNosy & The Debugger**
- Work with both MPW 3.0 and 3.1 SYM file formats
- Cmd-shift-Minus of a decimal number will convert it to Hexadecimal
- Add AND, OR, NOT and DIV to the operators in the expression parser.
- Cmd-period to cancel a dialog or to abort reading of the .SYM file, etc.

"Source Level Debugging" - Think's LightSpeedC Projects

For LightSpeedC projects compiled with "**Use Debugger" on**, *The Debugger* will display the source code of the procedures in windows. One may optionally toggle between source and interspersed ASM with a Command-Click in the content region of the window. The types of the global variables may be supplied by the user in a ".dsi" file. The names of the local variables may be inferred from the ASM display.

Oops, The following didn't make it into the Manual:
To pass parameters to a "?UserP", use assignments to global variables in the "action clause" that calls it.

The Scroll bar in the "**Type@nnn**" windows is a "page" type scroll bar that does a page up/down when clicked in the gray region. It is useful in the case you have an array or record. An example is to define the type:
`trapTbl = record a,b,c,d:Array[0..3] of ProcPtr end;` and do a `"trapTbl@400"`.

The syntax of a command in Nosy is: **Command_word** followed by **arguments** (if any) followed by a **return.**
For a given command, only the letters that are **CAPITALIZED need** be entered.
Arguments enclosed in angle brackets (<>) are mandatory. Arguments enclosed in curley braces ({}) are optional.
A vertical bar (|) is used to note alternate options (a|b = a or b). You may enter spaces only in numeric fields.
A command with a **leading space** will be treated as a comment in the main menu,and ignored in other menus.

As an example consider the command: "Review{Untyp|All}{d#}"

Typing in an R defaults the command to "RU1" (review data of the untyped blocks starting at "data1").

When a command is writing output to the console, it may be temporarly halted by clicking the Mouse button. Press the **space bar** to enter the intermediate menu, where you may terminate the command by typing Quit (q followed by a return).

Listing Format

The format of lines of instructions displayed on the console is:

rrr: hhhh xxxx label opcode address ; comment

where:

rrr - segment relative address
hhh - hex of the instruction (may be truncated)
xxx- is either the ascii value in sngle quotes, or the address of a jump instruction ($SSrrrrrr) or the offset of memory reference
Label, opcode and address fields have the obvious meanings.
SSrrr - top 2 digits are the segment number and low 6 are the segment relative address

Design Goal

Nosy is more of a decompiler than a disassembler. Its purpose is the recovery of source code in a human readable format that reflects the intent of the code's author. In order to achieve this goal it treats the program it is disassembling as a tree of procedures. When one ignores recursion, as I do, the program Call Graph reduces to a DAG (Directed Acyclic graph), which for the sake of brevity, we will refer to as a tree.
Nosy does a tree walk to locate the regions that are code. The remaining regions are marked as data. At present it does not gather enough information to determine which data areas are really code (procedures passed as address's) and human help is needed to supply this information via the Review data and By _attributes commands.
One then repeats the tree search (Explore command) and continues until the program is in its original format, or some reasonable representation of it. Data formatting facilities (Review data command) are provided to adjust the presentation of the data areas to that which the author intended.

Concepts - Blocks and Reviewing Data

A block is a contiguous set of bytes identified by a first byte address and a length which may be zero or negative. Blocks are used to identify code and data areas. Data areas are displayed with a negative length in the blocks table. Code blocks with a zero length are "contained" in the first preceding code block with a non-zero length, and will be listed with their container. The output of the tree search (explore) is a set of blocks identified as code or data (the Blocks command in the Tables menu), a set of symbol tables, some miscellaneous tables for case statements, etc.

References to addresses are divided up into the following classes:

proc - symbols referenced by a JSR or BSR instruction
lxx_ - local labels within a procedure (where x ranges from a to z).
glob - symbols referenced as d(A5) (the global variables)
data - referenced by a LEA/PEA or not "reached" as code in the tree walk.
com_ - symbols for blocks of code reached by a jump instruction (Bxx, BRA, JMP) that are not "properly" in a procedure.Their existence is indicative of "common" code.

The following symbols may appear in a procedure that starts with a LINK A6 instruction:

vxx_ - the local variables (vxx_1 is the offset relative to of the first local variable)
param"k" - the offset of the "k"th parameter relative to the stack frame.
funRslt - the offset of the function result

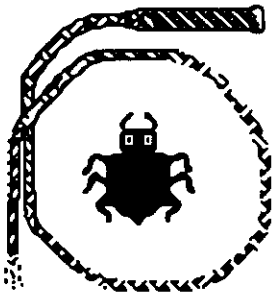
Nosy obtains the names of the Mac trap routines and system global symbols from its interna tables. It will pickup the names of routines left in the code (at the end of procedures) by Lisa Pascal, or read in the entry point names on a **.MAP** file created by MDS or the Consulair Linker. After looking at a procedure to see what it does you may use the **Change** command to rename the proc,global, data and common labels.

While **Reviewing data** one may change the formats of data blocks. The possibilities are -
-**U**ndo - change the block back to its original size and format
-c**O**mbine - if prev blk is data blk then combine it & current into 1 block
-**H**<B|W|L>n - print block as Hex Bytes, Words or Long words with "n" items/line.
-**D**<B|W|L>n - print block as Decimal values
-**A**<B|W|L>n - print block as Ascii characters
-**W**str,**S**tr - print block as a Pascal String, W prefix for word aligned
-**WZSTR**,**Z**str - print block as a Zero byte terminated string, W for word aligned
-**J**umpc - block is a set of jump table entries relative to beginnning of block

Debugger Tech Note #10

Debugger Manual Changes

July 91



Information

Control

The Debugger manual was reved in July 91, pages 33 -34 and pages 63 - 64 were changed as follows:

Pages 33 - 34 :
The description of the .dsi file commands were revised to clarify:
a) That any lines following the '=End' line would NOT be processed. I.e. they will be treated as comments.
b) The =ROMI section now is:

=ROMI
followed by lines with:
1) the addresses of Trap calls for Discipline to ignore. You obtain the address's by selecting the contents of the "**-Show Ignores-**" Window and pasting it into a ".dsi" file
2) a typeName prefixed by a minus which you wish to omit Discipline for (-Char, ...)
3) a "_Trapname' prefixed by a minus which you wish to omit Discipline for (-_Open)

Pages 63 - 64
The Bondage menu was revised to move the set/clearing of Trap Discipline and its options into a dialog box.

Trap Discipline ... - Displays a modal dialog that lets you activate discipline and options associated with it. The three indented options, are only active when Discipline is selected. They are:

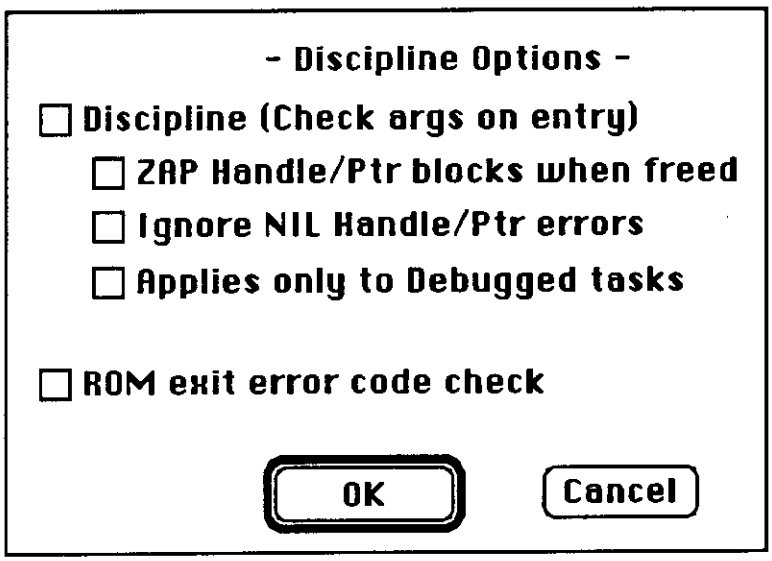
Zap Handles - Toggles "presetting" of memory. When selected, blocks (in the heap) released by a _DisposHandle or _DisposPtr call will be preset to a garbage value so that any references to it may result in address errors. In addition, if the handle/ptr to the block being released is stored in a global variable, then that variable is also set to garbage so that future references to it will cause an error. The present value of the preset is "$fffD fffD", this value will cause Bus Errors on the 68020.

When the **Ignore NIL Handle/Ptr Errors** box is checked NIL handle/ptr errors will not be reported.

When the **Applies only to Debugged tasks** box is selected, errors will be reported only for those tasks that have an Aux file or are a Think C project.

ROM Exit code check - Toggles checking of the results of Trap calls on exit. Trap calls which set error flags (**MemErr** or **ResErr**) or return an error code of type OSErr are checked, and will cause a break into ***The Debugger*** if they are returning an Error Code (a negative number). This option may be selected independent of Discipline.

NOTE: Rom Exit code checking DISTURBS the execution of ROM patches with "Come From" code.



-**LADRC** - block is a set of DC.L com_xxx
-**LADRP** - block is a set of DC.L PROCxxx
-**BRAC** - block is a set of BRA.L com_xxx (use the D command in the intermediate menu to tie it to a JMP)
-**BRAL** - block is a set of BRA.L loc_xxx (use the D command in the intermediate menu to tie it to a JMP)
-Code - change block to code
-**DRVHD**=label - Treat the current block that is 9 words long as a Driver Header, i.e. 4 words of binary info followed by a 5 word jump tbl. Prefix the labels in the jump table with "label". See the Serial drvr jrnl files on Delphi for useage.
-**New**{Bytes} cnt - Split the block up into 2 blocks. Cnt is the number of words or bytes (decimal) in the first of the 2 blocks. This command is used to split up blocks that are mixed data and code.
-**NewUntil**{hhhh} - split the block up into 2 blocks as follows: search the current block for an occurence of "hhhh" on a word boundary. The block is then split at the word following. The default search is for a RTS, RTD, RTE,RTR or JMP (A0) (hex values 4E7x, x = 3,4,5,7 or 4ED0). If hhhh is present, the last four hexits entered are used for the pattern. If it is not found, then an error is returned. For example, "NU" splits a block up after the first RTS instruction. **In Version 2, the block is automatically changed to Code**
-**NewLast** hhhh - split the block up into 2 blocks as follows: hhhhhh specifies the byte address of the last word in the first block. For example; NL44 will cause the second block to start at 46.
New* - split the block up into 2 blocks by using the byte count in the first 4 byte of the block as the new length. Ignore garbage in the upper 6 bits.
-<H|D><Fld>cnt - defines a bit field that is cnt bits long. H|D is display fmt. **User beware!**
-Mname=fmt1,fmt2,..., - define the block to have the repeating sequence of formats. For example the STRG rsrc in the Mac editor may be formatted by the definition "Midstr=db1,db1,Zstr" or Midstr=db1,Str,zb1" . The command jump table in Macsbug may be formatted as "Mcmdtbl=ab2,jumpc=cmd_+p1" and the opnib jump table as "Mopnib=jumpc=opn_+0". The syntax is jumpc|addr = <label_prefix>+<suffix> where suffix may be Pn for the n'th parameter in the macro def or n which defines the base of the labels (opn_0, opn_1,...).
-=mac_name - the previously defined macro mac_name is used for the block being reviewed.

After a block is changed to Code one has the following options -

-Undo - change the block back to data (original size and format)
Revert_to_data - change the block back to data , but retain current size
Is_proc - Tell nosy to remember that this is a code block for future tree walks (Explore command). Note that if a block contains multiple independent procedures, then only the first will be remembered. Use the Revert and NU, etc commands to split the block up to minimize multiple explores.
-New... - Split the block up into 2 blocks. See above for description.
-Back_to_base_blk - if you have split the previous block and changed the current to Code, then the effect of this command is to restore conditions to that before the split.

To change a block back to data after it has been changed to code, "Is_proc" and you have done a treewalk, use the Not_proc command in the intermediate menu

Strategy (for using Nosy)

Now that you know what some of the formatting commands are, I will now suggest a strategy for recovering the code. After the initial tree walk do a review data to change blocks

Real Soon Now at your Local Macintosh CoverTest - A Code Coverage Analyzer from Jasik Designs

What is CoverTest?

CoverTest is an application designed to help programmers and Quality Assurance testers determine which parts of a program have been executed.

Features

• Automatically extracts symbol information from MPW SYM and Think C™ project files.

• Instant access to the source code of a procedure with vertical bars noting which lines have NOT been executed (see below window).

• Dynamic updating of the test results allows one to adjust his path through the program to excersize partially tested modules.

• Merge test results from different runs/testers.

• Selective filtering of the results.

• Sort results by name or percent executed.

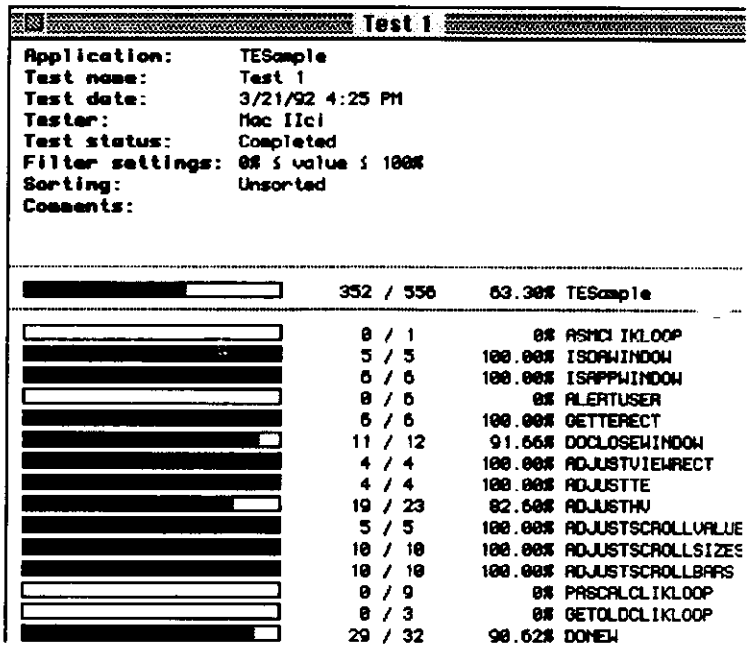
• Option to export results in a tab delimeted file for use in a spreadsheet program.

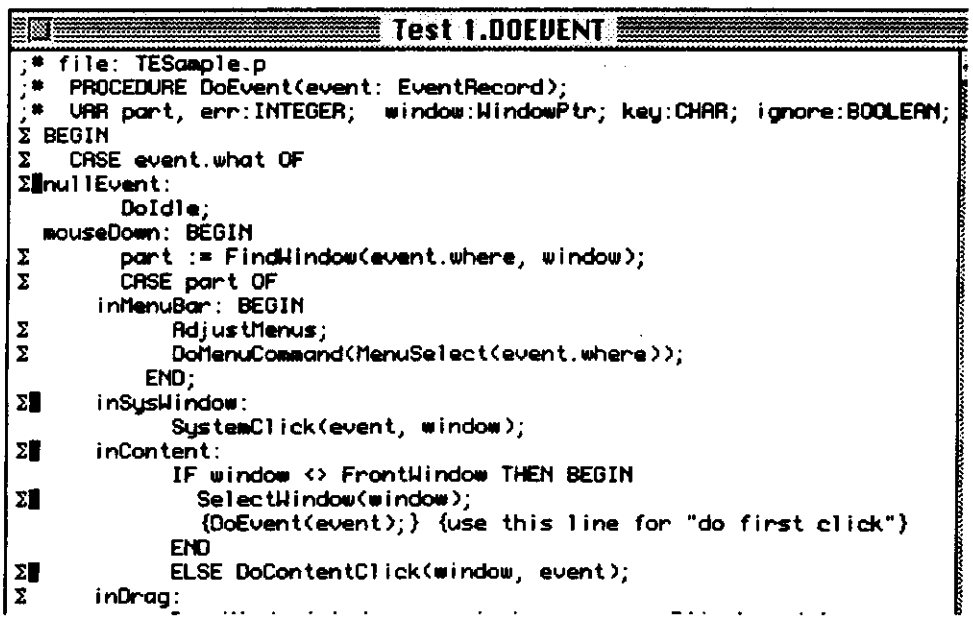
Availability

CoverTest will be bundled with The Debugger and will be shipped to current owners of it in June 92. An unbundled version that runs independently of The Debugger may be released in August.

Requirements

A 68020/030/040 Macintosh with > 4 Mb of RAM





In the above, the columns are: The bar graph, showing percent executed; the number of statements executed; the total number of statements; the percentage executed and the procedure name. The first line is for the Application, and the others are for the procedures.

To the left is a source window. Notice that lines that have NOT been executed are marked with a ▮ character.

Main menu commands

-Review{Untyp|All}{d#} - review data blocks starting with d# (1 if absent) and set their format or change them to code. Some of the formatting has been done automatically by Explore. Reviewing only the untyped blocks is the default.
-By_attr - list the problem procedures (those that contain constructs that Nosyr cannot resolve without help
i.e. JSR (A0), traps, reset and some JMP (Ai) instructions). These are called mystJMP, mystCase, MystJSR, etc where myst is short for mystery. To search for mystery JMP's enter the number below the label mystJMP and a carriage return. At present you may search for any combination of things, but it may get a bit confusing.
-Explore - reExplore the resource, given additional info that the has been supplied to Nosy (ls_proc info, etc) by you.
-Display - takes you into window mode r where you can view and edit files.
-Allproc{seg#|Bblk#| Bn-Bm} - list the resource starting at seg# or blk#. "A" causes all segments to be listed.
AB20 causes all blocks starting from 20 to be listed, check the Blocks listing for a starting block number.
-Proc_byname - list a proc and the procs it calls in call order. In response to the prompt "proc name|#|=str|=addr" , a "5"
will cause proc5 to be listed, "=event" will cause the listing of all procs with the string "event" in their name,
=addr will list the proc that contains addr. For example when looking at ROM, =400420 will list the proc containing that address.
-addr - starting at a given address list 500 bytes as code. Entering 0 (zero) will cause the entire file to be listed as code (ignoring the results of the tree walk). Note that Nosy makes a prepass over the file build a local label table so it may take some time before it starts listing a long file.
-Hardcopy - opens/closes a disk file for output from the A,P,B,S commands. The syntax of the command is: H{volumename:}{filename}
"H" results in output to "filename.list" or "filename.asm".
"H?:" lets you specify the volume name bv presenting a special display. Mount the volume you want output to go to and make it the default volume with the set_Default command.
"Hmydisk:" results in output to "mydisk:filename.list". "Hxxx:myfile." results in output to "xxx:myfile"
"+" before Hardcopy command in the menu tells you that it is 'on (file is open)'.
-Jrnl - toggles the journaling flag, initially on. A "+" before Journal tells you that it is 'on'.
-Wait - a nop when entered from the console, but when encountered on a jrnl file causes Nosy to wait for you to press the space bar.
-Tbls - brings up Nosy's debugging menu - the Block list should make sense to most programmers
List_rsrc type<id> - lists the resource map or an individual resource. examples are Lcode, Lcode0, Lpack, L, etc.
-ref_Map - takes you to the Map menu where you can find out what proc,data,glob,com labels, system symbols or trapnames are referenced.
-Outredir{filename} - redirect all console output to filename. Use the same syntax as the "H" command to specify an alternate volume.
-saV_snt - saves Nosy's internal tables to the file "xxx.snt"
-Quit - exit the main menu to save the files, and select a new file to disassemble or exit Nosy.

New Features and Fixes, etc (see the "Change History" file for a complete list)

- Minor changes to CoverTest - ability to select all & copy from a text window, etc.
- Replaced the internal text editor in both The Debugger & Nosy with one that supports:
  More than 64K of text, tabs and Undo/Redo. It gets the tab size from MPW's 'MPSR' 1005 resource.
  The editor is RAM based so you may want to increase the Debugger heap size to 800K or 900K.
- Full name demangling for C++ names in The Debugger.
- Removed '.dpf' file processing (used by IBS for Pascal 3.1).
- The leading char string search used by **Cmd-D** has been generalized to any string that you type in. If the entered string matches only one name, then the procedure is displayed, otherwise a list of all proc names with the same prefix is displayed (Debugger only).

fix - ("Safe SYM") The 7/26 fix to the handling of VOID type defs produced by C++ was incomplete and caused infinite loops in Dbgr when one attempted to display type@nnn.
Void types will display as "???" in the type definition, and the value will be shown as a Hex Long.
fix - Miscellaneous blowups, etc. on 'Step Into' a proc due to incorrect calculation of the caller's address.
fix - Spurious "Perf Time on at Tool Exit" msgs when Shell & ToolServer are active.
fix - Incorrect Heap check error msgs when inside a _MoveHHi call.
fix - Incorrect display of register type of Vars assigned to FP regs
fix - Rewrote Bkpt processing to avoid disk bashing for segments NOT yet loaded.
fix - Nosy has to force the stack size to $6000 so treewalk works on the Mac + & Classic.

- Dbgr is configurable for Foreign keyboards etc. in ResEdit using the 'Dcfg' resource.
- Shift-'Heap Scramble' scrambles the System Heap (no auto turnoff)
- Added a Performance Timing command & windows (see DTN #10 for details).
- Added ability read protect 0 page of memory in MMU Protection.
- Heap display - move summary to front & display Detached Resources (file refnum = '<??>').
- Added Dbgr flag 'Sgned_Char_Strs' to force 'Array of char' to be a string.

The Debugger is a Macintosh Application ??

In order to maintain uniform behavior of The Debugger over a variety of ROM versions, I have licensed and incorporated private versions of the Event, Window and Menu manager traps that one normally calls. The consequences of this and other things that I do are that The Debugger will no longer be affected by INITs such as QuickKeys and most of the disk doubler procs. Your source files must be uncompressed to view them.

Known Problems

- **The Debugger does not work with System 7 VM**
- The text drawing routines put garbage on the user's screen when using Step Continuous.
- The Debugger and some disk drivers (Appolonius, ...) do NOT get along, so you will have to use a different driver such as SilverLining or HDT, etc.
- Syquests must be mounted at Boot time in order for The Debugger to "see them".
- Do not try to use a Syquest with the R45 INIT as a boot disk because The Debugger will screw up.
- When doing a Shutdown on some PowerBook 170's one gets a spurious BusError in The Debugger.
- Attempting to set a breakpoint at an address > 16Meg that is NOT in a task will not work.
- Using MMU protection with programs built with Model FAR causes spurious errors, so move the CODE segment named '32-bit bootstrap' to the System heap to avoid them. See Dbgr Tech Note #7
- Using the Think C Debugger in combination with The Debugger causes programs to stop at the end of The Debugger's patch to _LoadSeg because the TC Debugger sets `LoadTrap` (byte at $12D).
- The MDEF distributed with IBS for use in the MPW Shell **does not** work with System 7.
- MMU protection does NOT work on Mac II's with the Dove Accelerator installed.
- The Debugger does not handle .SYM file info for arrays with variable bounds emitted by LS Fortran.

Future (Work in Progress for the Dec 92/Jan 93 release)

- MMU Protection for Macs with 040 CPUs
- PatchLink loses the type of local variables whose types have no name (^^WindowRecord, etc).

Sincerely,

Steve Jasik **AppleLink: D1037** **Compuserve ID: 76004,2067**

Search Commands (main menu)

-SAddr_ref name_list - search code blks for given address
sap2,g4 - search for refs to proc2 and glob4
-STrap_ref name_list - search for references to given trap names
stopen-launch - search for refs in the range open to Launch
-SRsrc_type char4 - search for refs to 4 char strings
Rcdef - search for refs to 'CDEF'
-SString chars- search the file for refs to an arbitrary ascii string.
ssppl -to find the copyright notices in ROM. Note that this command is case sensitive ! !
-SHex hhhh - search the file for refs to a string of hex digits. The string should contain an even number of hexits.
-SMark - search the data areas of the resource for data strings that look like disk data/address prologue (D5 xx AD) or epilogue (DE AA FF) markers. The copy that is referencing these markers is probably doing funny things to thedisk. The SM command is a quick way to find the root of copy protection code for many programs.

NOTE: that the capabilities of the Search and Map commands overlap. The SA and ST commands will not find anything that the Map command will list unless you have done a Review_data and changed some data blocks to code and not done an Explore. In this case the SA, ST commands will be more accurate.

Display command and Window mode

What started out a simple text editor has now acquired additional menu and capabilities so one can do a fair amount of work in window mode. Desk accessories are supported in window mode, and a separate help file is available for it. The Display Menu contains commands to display Procedures, procedures names, the references to a name, the system sysmbols referenced by the program, etc. For complete information consult the Help facility in the mode. Note that the Help file is a text file and may be printed by any text editor.

The Intermediate menu commands

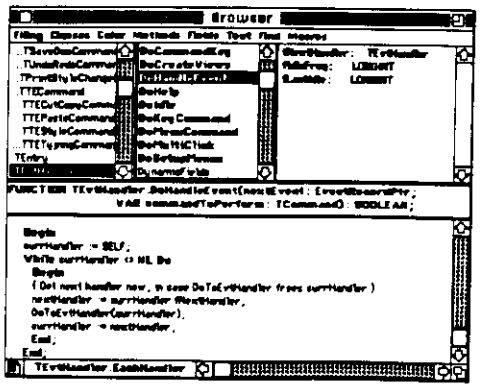
This menu is encountered when one presses the space bar during the listing of a code block etc

<cr>= eXit_cmds
-conVert{addr} - converts a program address (seg# and seg rel addr into a file relative address, if "addr" is ommited, then Nosy uses the last listed address.
-Change P|D|G|C #/newname - change symbol name to newname
"CP2/myname" changes "proc2" to myname.
-Is_proc <data#> - See discussion of data formatting above.
-Define_case_jmp jmp_addr - Fill in information about a JMP table that Nosy couldn't figure out.
-User_stop - When inserted in the journal file by Nosy this will cause the code block listing to stop at the same place. Useful for educational purposes, etc.
-Tbls -bring the table dump menu.
-Quit - Terminate the current command and return to the main menu.
-Not_proc name - Changes name from a proc to a data block and removes the Isproc entry for it.

OBJECT MASTER

Browser Windows

Class hierarchies can be viewed either in outline form or via a graphical hierarchy. You can view classes details by double-clicking on a class, revealing its header information, methods, field types, and inheritance information. All editing functions can be performed from the Editing Area within a Browser window. The number of Browser Windows that can be open is limited only by the amount of available memory.

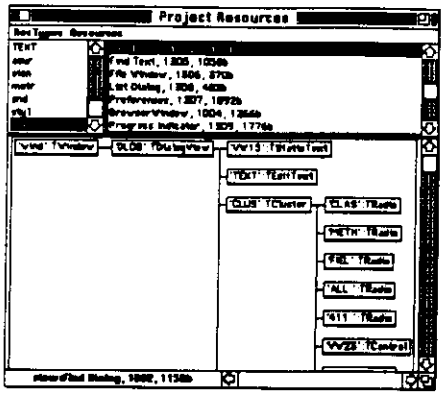


The Browser Window combines easy navigation with full source code editing facilities. You can open as many Browser Windows as memory allows.

Segmentation, File, and Resource Mapping

Segmentation and File Lists provide a quick way of viewing the segmentation plan for a given project. You can view resources from within Object Master.

If you wish to edit them, double-clicking on one will launch ResEdit and open the appropriate resource. You can customize your resource view by resource type allowing you to work with resources in the way best suited to your application.



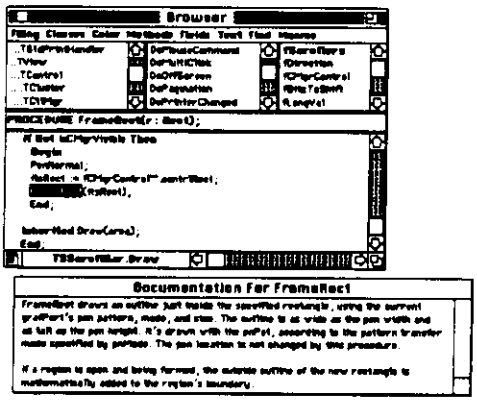
Object Master gives the programmer customizable resource previews specifically designed for programming reference. ResEdit (or other resource editors) can be automatically opened for editing directly from the Resource Preview.

Interactive compiling

Projects can be compiled in the background with the MPW ToolServer. A list of errors is returned to Object Master and displayed in a scrolling window. When you click on an error, Object Master takes you to the line of code that generated the error.

On-line documentation

Access is provided on-line to 411 (Inside Macintosh) documentation. You can also automatically generate your own 411 (help) file from your source code.



Object Master provides full access to all 411 documentation. In addition to using the 411 files provided by Apple, Object Master can automatically create 411 from your own source code.

OBJECT MASTER was written by Loïc Vandereyken
OBJECT MASTER is a trademark of ACI, ACIUS, Inc.

Worldwide distribution: ACI, Paris, France: 33-1-42-27-37-25
United States Distribution: ACIUS, Inc.

ACIUS, Inc.
10351 Bubb Road
Cupertino, CA 95014
(408) 252-4444

ACIUS

Map Menu commands

-All_refs {G|D|P|C|T|S} - Lists all the references to the sets of symbols. agp - lists all the refs to the globals and the procs. a - lists everything, at - list trapnames, as - list system syms
-Refs<name_list> - same syntax as the ST command. Rc1,p1-p5,t0 will list the references to com_1, proc1 to proc5 and Trap 0 (Open)
-Fmt_by<Proc|Addr|Name> - Format the reference list by Procname, Address (seg#/seg rel addr) or Name (procname+proc_rel_addr)
-Seg_map - produce a map of the block names (procs and data). The format is:

SEG#] procname first_byte_addr Length reflist

The ref list is in the format "n/procname", where n is SEG# that the proc is in if it differs from the current seg. Useful for "balancing" code segments.
-Trace <C|P #> - Lists the procedures that call a given proc|com. Similar to a static 'stack crawl'. Use it in conjunction with the **SM** command.

Table dmp and debugging menu commands

-Blks{S#|addr} - list the blocks table starting from blocks in segment S# or blocks greater than "addr". Used to check the overall structure of the program, or to find out the block # of a proc. BS2 lists all the blocks starting at segment 2.
-Symbols{wf} - list the names and values of the canned sym tables. If wf is present, then it writes out files of EQU's.
-Tbl<G|L|D|C|P> - lists the address's of the symbols in the various tables
-Heap - lists the current state of Nosy's heap
-Names - lists the names in Nosy's name table
-Mem<addr/len> - lists memory in ascii and hex starting at "addr"

Class Library (continued)

OOPC offers unparalleled event processing capability. OOPC has extensible, multiple-priority event handling. Operating system events are at one level. There is a higher level (ASAP), and lower level event queues. There is also a timer event queue for periodically repeating events. You can create new event queues at any level you need. Also, events at any level can be scripted.

OOPC comes with object persistence built in. Adding file operations to new classes requires little or no new code.

Great Debugging Tools

The most important debugging tool is a way to trace program execution, so you can follow what's going on. OOPC can automatically create an execution trace file for you. OOPC also includes an object browser that hooks into the THINK C Debugger: you can see an object in its own data window with a single click in the Debugger data window. And the manual includes a section on debugging tips and techniques.

A Comprehensive Manual

OOPC comes with an easy-to-read 420-page manual that explains everything you need to know about programming the Macintosh using OOPC. The manual clearly marks important technical tips and reference sections, so reading and referencing the manual are even easier. The index has over 2000 entries, making it easy to refer back quickly to the topics you want to review.

Hit Bedrock with OOPC™

OOPC is designed for portability. The Macintosh version is now shipping. We're already working on the next version, and it's not just for the Mac.

INDUSTRIAL STRENGTH SOFTWARE™

Comparison of OOPC and C++

Feature	*OOPC*	*C++*
Multiple Inheritance	Yes	Yes
Class Variables	Yes	Yes
Class Methods	Yes	Yes
Multi-Methods	Yes	Yes
Object Methods	Yes	No
Multiple Dispatch	Yes	No
Dispatch Control	Yes	No
Dispatch Optimization	Yes	No
OODB Support	Yes	No
Dynamic Definition	Yes	No

Blazing Memory Management

OOPC memory management has been clocked at ***over fifty (50) times faster*** than Macintosh OS memory management. This is because OOPC memory management is optimized for handling a large number of blocks, while the Mac memory manager bogs down as more blocks are allocated. All object-oriented programs use thousands of blocks, so any object-oriented development tool that uses Macintosh memory management suffers a heavy toll. OOPC speeds on by.

Use THINK C or MPW C

You can use OOPC with THINK C or MPW C to develop Macintosh applications faster than ever before.

The Bottom Line

There is much more to OOPC: the consistent use of simple verb functions, such as act and draw, for all class methods; simply great exception handling; automatic performance profiling. The feature list goes on and on. But what's important is that this is all ***harnessed power: easy to learn and simple to use, but when you need more, the depth is there.***

We want OOPC to work for you. As a software developer, you know there's ultimately only one way to be sure that you can track down and fix problems quickly: you've got to be able to get to the source. You're in control, because you get ***100% full source code.***

Trick or Treat: $299 Introductory Price Expires Halloween

rsion 1.1
hipping
$350

Electron Mining™

Better Living Through Software™

718 Bounty Drive, Suite 1819, Foster City, CA 94404 **800-453-1131** voice / fax (415) 341-2400

Removing Copy Protection from programs (Cracking 101)

At present most commercial firms are using the "key-disk" method of protecting their products from unauthorized copying/backup - The firm duplicating the disk puts a file with a "bad" track on the disk that the normal Mac utilities cannot reproduce. The program checks for the existence of this funny file during program initialization and if something is amiss then it refuses to run or mysteriously blows up, etc. Many games are protected by elaborate protection schemes which involve encrypting part of the program, using non-standard resource ID's for the CODE resources, putting garbage in CODE0, etc. They are difficult to crack without first removing the encrypted parts etc.
For "normal" programs , an algorithm for finding and removing copy protection is as follows -

Make a backup of the program with one of the nibble copiers such as Copy II Mac (sector copy) and run it to find out what nasty things the programmers have taught it. Apply Nosy to the program. After the explore phase try the SM command to find the funny call to the Sony Drivers. If it finds anything of interest, then use the Trace command from the Map menu to find the chain of routines that referenced the data block. Jot down the names of the routines, and use the P command to selectively list them. By inspecting them you should be able to decide which routine to patch to bypass the copy protection algorithm. When inspecting procedures with the P command you will encounter numerous short "glue" routines which contain only 1 trap call. Use the Change command to name them with the same name as the trap call so that when you go back and relist the calling proc you can see what it does.

The following example illustrates their use in ROM to find the standard values for the address/data markers:

```
?SM

searching for disk data/addr markers ($D5 xx AD/96)  (example uses the 64K ROM)

match at addr =  402070

                                                          ;-refs - proc164
402070: '...'          data29          DC.B   $D5,$AA,$96  ;default values
402073: '...'                          DC.B   $DE,$AA,$FF
continue [y] ?n

?Map

Trace<C|P #>

>p164
                       proc            called by
  402096               proc164         proc115
  40195C               proc115
```

The above method works for MacDraft, MacPascal, PageMaker, and many other popular programs.

If the SM finds nothing of interest then check the reference map (AS) for references to system symbols. If still nothing, then do a Review data to normalize the proc names and look for dumb messages such as "program disk must be in internal drive" (ThinkTank128). Also look at the DITL's which may contain dumb messages. In the case where such a message exists the program referencing it is displayed by Review data right above the string containing the message. Inspect that procedure with the P command to find the decision point and try changing it to a branch unconditional (60xx)or set of Nop's (4E71) as necessary.

For MacPascal use the SM command, or check the refs to ApplZone. One may have to plug a bit more code since there are 3 magic files that it checks along with a watchdog routine to

• Cmd-D - if a partial name (Class::mm) is typed in, then open up the procedure that contains that name OR list all proc names that 'match' in a window.
For example, if `Txyz` is a class, then entering "`txyz::`" into the Cmd-D box will list all its methods.

New Features and Fixes, etc (see the "Change History" file for a complete list)
• MMU protection in 32 bit mode for 030 CPU's (IIci, IIfx, IIsi, ...)
• PatchLink - globals seg may exced 64K. Add '-far' option to force correct map file processing for programs that are not built with MacApp.
• expand 'ShutDown Taskname' so it works properly with MPW Tools
•new Heap Scramble is FAST, task rel, works in 32 bit mode & turns itself off when the task ends.
• if 'Region_@nnnn' is Hilited when "`" key is pressed to view user's screen, then the selected Region will be Inverted. Useful in viewing the UpdateRgn , etc in a WindowRecord
• Revise handling of C++ types to automatically compensate for the incorrect type info from CFront. i.e. - do an extra deref automatically for Txxx@nnn if nnn is a MasterPtr, etc.
fix - Find/Change dialogs were ignoring the Token & case sensitive check boxes and the presence of a Find dialog would interfere with Cmd-G, etc. Allow Files:Close to close the Find/Change window
fix - error -36 text - it is a Hardware I/O error such as a Disk R/W error
fix - distinguish between NIL = 0 and 'NIL - NOT Init' = FFFDFFFD pointers
fix - blowup in SYM file type processing when more than 8191 types on input

Known Problems
• The Debugger & Radius RocketWare V1.3 do not get along, contact Radius for a newer version.
• The text drawing routines put garbage on the user's screen when using Step Continuous.
• When doing a Shutdown on some PowerBook 170's one gets a spurrious BusError in The Debugger.
• Attempting to set a breakpoint at an address > 16Meg that is not in a task will not work.
• Using MMU protection with programs built with Model FAR causes spurrious errors, move the CODE seg named '32-bit bootstrap' to the System heap to avoid them. Use the following Shell command:

```
echo "change 'CODE'('32-bit bootstrap') to $$type ($$id,$$Attributes|sysheap );" ∂
  | rez -a -o myAPPL
```

• MPW C outputs bad SYM info for enum declarations that contain refs to symbols previously defined in the enum (enum {a,,b, c = a | b};). The Debugger may blowup during type processing. A fixed C compiler should be available by ETO #8 time. QuickTime users should fix Movies.h.
• Using the Think C Debugger in combination with The Debugger causes programs to stop at the end of The Debugger's patch to _LoadSeg as the TC DEbugger sets `LoadTrap` (byte at $12D).
• The MDEF distributed with IBS for use in the MPW Shell **does not** work with System 7, do not use it.
• **The Debugger does not work with System 7 VM**
• Do not try and use a Syquest with the R45 INIT as a boot disk, The Debugger will screw up.
• MMU protection does NOT work on Mac II's with the Dove Accelerator installed.
• The Debugger does not handle .SYM file info for arrays with variable bounds.emitted by LS Fortran.

Future (Work in Progress)
We will be working on the following features for the next release (late May or June 1992):
• Covertest, the code coverage analyzer.
• Enhance MMU protection to detect reads of location 0.
• Enhance expression parser to allow writes to alternate windows, loops, etc.

Debugger Tech Notes (DTN's) are on the floppy in SuperGlue Viewer format, you may view/print them from AppleLink or SuperGlue. DTN's 0, 5, 6, 7 and 9 have been revised.

Shows I will Attend
I will be at the WWDC in May and the Boston Macworld in August (at the Apple Tools expo).

The Head Nose on CD-ROM/QuickTime
Will be mailed out in June/July. All current registered users of The Debugger will receive it.

Sincerely,

Steve Jasik **AppleLink: D1037** **Compuserve ID: 76004,2067**

make sure that you have not patched the file check routines. In most cases the above works because the magic files do not contain any information is necessary for the execution of the program. Once you have cracked a program either the cracking information or the journal files can be distributed. To find the checking code without wasting a lot of time search for references to the rom tag byte ($400009) which is used to decide whether the program is running on a Mac or a Lisa under MacWorks or the Lisa Disk address's (SA0FCC01E) or Open trap calls. Having found one of the above use the ref map or the search command to walk up and down the tree until you find the offending code. A word about Microsoft products. They use a UCSD style P code interpretor and all the interesting program code is encoded in that format. I havn't had time yet to decipher it or to write a disassembler for it. (***Macintouch*** is an excellent Mac Newsletter which has a good article on copy protection in the July 85 issue. A subscription is $48 per year. Ford-LePage Inc. P.O. Box 786, Framingham, MA 01701 (617-527-5808).)

Suspend/Resume

Suspend and resume are not commands, but queries at the beginning and end of a disassembly of a file. They should be used for **files that do not change** such as the Rom, MacPaint, etc. The suspend files are named "filename/rsrctype.snt". The "snt" suffix stands for "saved Nosy tables". The files contain enough info to bypass the tree walk, but not enough to check for different versions of a file, etc so try not to abuse the feature. Those who wish to do further analysis of the program structure can use the tables in the file as they wish.

Jrnl files

Journal files are the record of the command stream to Nosy. They consist of lines of text in the format "command | prompt", and you can edit them with Edit. Nosy has some simple synchronization logic so things don't out of kilter. Note also that Nosy doesn't record commands that are in error. Nosy reads the command line, appends the prompt and writes it out to a scratch file. At the end, one has the option to have the jrnl file rewritten with the current stream of commands. Jrnl files from multiple runs aren't concatenated by Nosy, you have to do it yourself. The jrnl file for Boot blocks uses FEDIT. A jrnl file for "ROM" is also supplied that illustrates block splitting techniques . The jrnl file "steal that Sort" is a simple one that illustrates the use of the Hardcopy command. The jrnl files indicate the techniques necessary to recover the source code.

ACI files and adding comments

Nosy V2 is capable of reading in a text file named "xxx.aci" and merging the comment lines in it into the disassembly of the file "xxx". The ".aci" file is read when the file is initially read in if a ".snt" file is present, and after an Explore. This is done to determine the byte address (for random reads) of the comment text for a procedure.

At present it accepts lines of the form: (<x> - mandatory, {x} - optional})

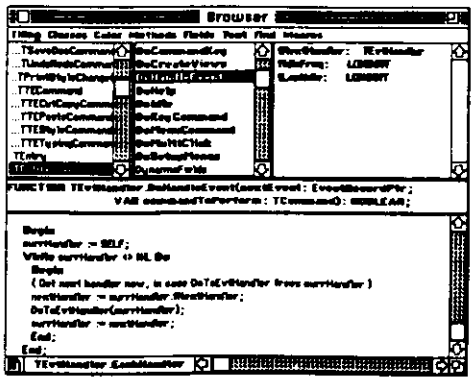
<address> is the segment relative address (leftmost columns of the listing produced by Nosy).

<address><=><Seg#></>{proc name} - Marks the start of a comment group for a proc
<address>< ><text> - comment line that is appended to line with <address>.
<address><i><text> - comment line that is inserted after line with <address>.
<address></><With structname{=An}> - Tell Nosy that An is the base register for references to a Pascal Record with "structname". The "=An" is optional when the instruction on the line defines An (i.e LEA xxxx,An, etc). /W lines may follow "i' lines.
<address></><Endwith structname> - terminate the range of a WITH directive.

OBJECT MASTER

Browser Windows

Class hierarchies can be viewed either in outline form or via a graphical hierarchy. You can view classes details by double-clicking on a class, revealing its header information, methods, field types, and inheritance information. All editing functions can be performed from the Editing Area within a Browser window. The number of Browser Windows that can be open is limited only by the amount of available memory.

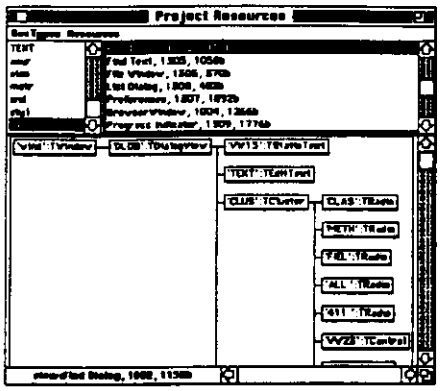


The Browser Window combines easy navigation with full source code editing facilities. You can open as many Browser Windows as memory allows.

Segmentation, File, and Resource Mapping

Segmentation and File Lists provide a quick way of viewing the segmentation plan for a given project. You can view resources from within Object Master.

If you wish to edit them, double-clicking on one will launch ResEdit and open the appropriate resource. You can customize your resource view by resource type allowing you to work with resources in the way best suited to your application.



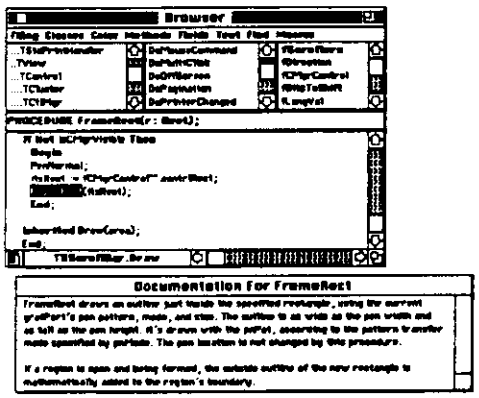
Object Master gives the programmer customizable resource previews specifically designed for programming reference. ResEdit (or other resource editors) can be automatically opened for editing directly from the Resource Preview.

Interactive compiling

Projects can be compiled in the background with the MPW ToolServer. A list of errors is returned to Object Master and displayed in a scrolling window. When you click on an error, Object Master takes you to the line of code that generated the error.

On-line documentation

Access is provided on-line to 411 (Inside Macintosh) documentation. You can also automatically generate your own 411 (help) file from your source code.



Object Master provides full access to all 411 documentation. In addition to using the 411 files provided by Apple, Object Master can automatically create 411 from your own source code.

Worldwide distribution: ACI, Paris, France: 33-1-42-27-37-25
United States Distribution: ACIUS, Inc.

ACIUS, Inc.
10351 Bubb Road
Cupertino, CA 95014
(408) 252-4444

Nosy does this automatically in most cases when the A register is redefined or at the end of the procedure. you will have to do it manually when a called proc redefines the register.

In any set of comment lines that apply to given address, insert lines may follow append comments, and "/" lines may follow insert lines. Comments placed before the first code line in a procedure will always appear after the "QUAL" line.

Notes on the new ROM

There are some strange uses in the the OS trap table. Note that in the symtem symbol definitions there are a number of symbols with walues between $0400 and $0800 which are prefixed with a "j'. These symbols are used internally by the ROM. Apple will document these uses sometime in Feb 86.

Bug Reporting

Bugs fall into the following categories

Critical - A blowup in the treewalk or incorrect disassembly of an instruction, etc that you cannot get around. They get first priority
Minor - Nosy is doing something wrong but it is just an annoyance
RSM (Requested Software Mod) - This is a request for a new feature in Nosy. This will be given consideration.

Bugs are reported by sending your **Labeled Nosy disk** with the failing program, and appropiate supporting materials such as Jrnl files, Snt files, and a description of the problem., to MacSerious Software, 36 Queen Street, Helensburgh, Scotland.

Versions of Nosy are marked by the build date in the title line (first line that Nosy outputs to the Console). The first V2 release is dated Jan 16,1986. The current release contains a number of new features and bug fixs that are described in the file **"Addenda"**.

• You may now **change command key equivalents** in The Debugger via the '-Menu' command in the .dsi file, see the documentation in ROM.dsi for details. Now you can close windows with Cmd-W !!

• IBS users - The C compiler has a -ONLY option similar to the one in the Pascal compiler, and it may work somewhat better than the one in Pascal. You can modify the PatchBuild file to take advantage of it. There is no -ONLY option for C++.

Quadra ROM disassembly

At the last moment I added some new 'Macros' to Nosy to recognize the location independent JMP and JSR instructions that are used in the beginning of the boot code.
The sequence : LEA absLong,Ai followed by a JMP or JSR dd(pc,Ai.L) is recognized as:
rJMP or rJSR procNN,Ai and the code at 'procNN' is properly explored.
To take advantage of all this you will probably want to start a ROM disassembly from scratch.
This code is of interest only to those who are building hardware and have to peek into the boot code, which is an unbelievable rat's nest of GOTO's and weird code.
For normal everyday running with The Debugger, use the supplied ROM '.snt' for the IIci...

New Features and Fixes, etc

For a complete list of the changes, consult the "Change History" file on the disk.

• 68040 support, and support for the 1 Meg ROMs, extensive testing on Quadra 900.
• Operates with AUX 3, no longer leaves dead cursors on the screen.
• Support for source level debugging of Think C Version 5 project files (many fixes in this release).
• fix - step into Think C Methods correctly
• fix - Debugger's patch to _SetTrapAddr so it works properly when another INIT patches it.
• fix - registers were destroyed when stepping into an unloaded segment.
• fix - replace 'snd ' & 'snth' resources in Dbgr with those from 7.0.1, do not unpatch _Sysbeep.
• fix to work with Model FAR, and also with MacApp 3.0 and Model FAR.
Note: you are limited to 64K as a maximum segment size if you want to use PatchLink.
• fixes to PatchLink and the IBS scripts to handle C++ properly, etc.
• fix - Methods Of ... (Ω-M) command to display C++ methods also
• revise the mods to MacApp so they work for MacApp V3. See the separate folder "MacApp 3 mods".

• For C++, Kludge the display of local Vars to transform '^^Txyz@nnn' into 'TXYZ@nnn' where Txyz is a C++ class name & TXYZ the corresponding Pascal Object name

• For C++ programs & Cmd-D - if a partial name (Class::mmm) is typed in, then open up the procedure that contains that name OR list all procedure names that 'match' in a window.
For example, if Txyz is a class, then entering "txyz::" into the Cmd-D box will list all its methods.
• Nosy - define default LIbs to search via a resource: 'NLib' , id = 0
• Added a special KLUDGEs to debug Multi-seg Print Drivers, call or write if you want more info.

Known Problems

• The MDEF distributed with IBS for use in the MPW Shell **does not** work with System 7, do not use it.
• **The Debugger does not yet work with System 7 VM**
• Do not try and use a Syquest with the R45 INIT as a boot disk, The Debugger will screw up.
• MMU protection does NOT work on Mac II's with the Dove Accelerator installed.
• The Debugger does not handle .SYM file info for arrays with variable bounds.emitted by LS Fortran.

Future (Work in Progress)

We will be working on the following features for the next release (Jan/Feb 1992):

• Enhance MMU protection to work in 32 bit mode, and on the 040 machines.
• Add the ability to target an arbitrary task so one can debug QuickTime components, etc.
• In pace memory modification (typeover) in hex-ascii & registers windows.
• Enhance expression parser to allow writes to alternate windows, loops, etc.

Sincerely,

Steve Jasik **AppleLink: D1037** **Compuserve ID: 76004,2067**

System Compatibility - Recommended System Level
This version of The Debugger and MacNosy has been tested on System 6.0.5, 6.0.7 and System 7. For the new machines (Classic, LC and IIsi) you must build a ROM/CODE.snt file from scratch.

New Features, etc
- Works better in 32 bit mode (miscellaneous fixes).
- fixed spurious error messages from The Debugger when heap check is on.
- Added Task Relative Discipline, and revisions to the bondage menu (DTN #10).
- IBS fixes to take advantage of the partial compilation in Pascal V3.2 on ETO #3.
- Operates with AUX 2.0.1 in 24 and possibly 32 bit mode (leaves dead cursors on the screen).
- Kludges have been worked out so that one can use the 8*24 GC card with System 6.0.x.
- Support for source level debugging of Think C Version 5 project files.
- 68040 support
- Some new debugger flags have been added (MB_DX, Del_Rsrcs, Slot_Mem).
- For C++ programs & Cmd-D - if a partial name (Class::mmm) is typed in, then open up the procedure that contains that name OR list all procedure names that 'match' in a window.

  For example, if `Txyz` is a class, then entering "`txyz::`" into the Cmd-D box will list all its methods.

Known Problems
- The MDEF distributed with IBS for use in the MPW Shell **does not** work with System 7, do not use it.
- Setting breakpoints in a .dsi file for a Think C project appears not to work (??).
- The Debugger does not yet work with System 7 VM
- Do not try and use a Syquest with the R45 INIT as a boot disk, The Debugger will screw up.
- MMU protection does NOT work on Mac II's with the Dove Accelerator installed.
- In 32 bit mode, the 'Use MMU' option does NOT implement MMU protection
- The Debugger does not work with MacApp when 'model FAR' is used.
- The Debugger does not handle .SYM file info for arrays with variable bounds.emitted by LS Fortran.

Future (Work in Progress)
We will be working on the following features for the next release (November 1991):
- In pace memory modification (typeover) in hex-ascii & registers windows.
- Enhance expression parser to allow writes to alternate windows, loops, etc.

Debugger Tech Notes (DTN's) on Disk
The Debugger Tech Notes are now distributed on disk in SuperGlue Viewer format, you may view them from SuperGlue or AppleLink.

The notorious ROM.snt message (excuses, excuses)
When The Debugger starts up, the '-Notes-' window contains the following lines:

```
reading ROM/CODE.snt
System PTCHs are different, you may wish to explore ROM
• addresses of ROM patches will NOT be adjusted
```

You should ignore the message. Both Nosy & The Debugger contain code that attempts to adjust the addresses of the RAM patches to the ROM so that they are correct relative to the current set of patches/INITs, etc. In most cases it is failing. The result of the failure is that if you attempt to display a RAM patch in The Debugger via Cmd-D that you may get a garbage window.

The Head Nose on Video
Will not be on a VHS tape, but on CD ROM instead. The reasons for the change are that with QuickTime, I think that I can get most of what I want to do on a CD ROM in a format that is universal for Mac programmers, and costs a lot less to distribute. The cost to duplicate & mail a VHS tape is about $10 compared to $3 for a CD ROM. I will be working on the tutorial this fall.

Sincerely,

Steve Jasik **AppleLink: D1037** **Compuserve ID: 76004,2067**